A numerical comparison of commonly - used algorithms for structural optimisation

Smith, Erling Aastrup

A NUMERICAL COMPARISON OF COMMONLY - USED ALGORITHMS FOR STRUCTURAL OPTIMISATION

by

ERLING AASTRUP SMITH

A thesis submitted to the Faculty of Science of Durham University at Durham in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

DEPARTMENT OF ENGINEERING SCIENCE
DURHAM

1975

ABSTRACT

SMITH, ERLING AASTRUP: A numerical comparison of commonly-used algorithms for structural optimisation. (Under the supervision of WILLIAM CALVIN CARPENTER)

The thesis makes a qualitative and a quantitative comparison of algorithms used to solve non-linear structural optimisation problems. Algorithms are categorised into linearization, feasible direction and transformation methods. From each category, algorithms are selected (by considering applicability restrictions, anticipated computational effectiveness and efficiency, supplementary program requirements and program development effort) for a numerical comparison of computational effort. The algorithms chosen are:- the Method of Approximate Programming, a Method of Feasible Directions and the Sequential Unconstrained Minimization Technique. Newton's, Fletcher-Powell's, Stewart's and Powell's methods are chosen for use with SUMT.

The algorithms are used in the study to minimize the weight of eight test structures:- four pin-jointed plane trusses and four plane stress plates, all subject to two load cases, member stress limits and design variable limits. The finite element stiffness method was used for structural analyses, function and derivative evaluations. Details and FORTRAN IV program listings are given for the algorithms.

Estimates are developed of the relative computational effort required by each algorithm in terms of the Central Processor Unit (CPU) time required when an IBM 360/67 computer is used. Measurements are reported for each algorithm of the CPU time used on an IBM 370/145 computer.

A comparison is made of the computational effort used by each algorithm. Conclusions are drawn about the relative efficiency of the optimisation algorithms and of the derivative algorithms.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

CPU	Central Processor Unit
FD	Finite Differences
FP	Fletcher-Powell's method used with SUMT
LP	Linear Programming
MAP	Method of Approximate Programming
MFD	Method of Feasible Directions
N1	Newton(1)'s method used with SUMT
N2	Newton(2)'s method used with SUMT
NLP	Non-Linear Programming
NMW	Near Minimum Weight
PO	Powell's method used with SUMT
ST	Stewart's method used with SUMT
SUMT	Sequential Unconstrained Minimization Technique
UOA	Unconstrained Optimisation Algorithm

NOTATION

a	scalar or integer subscript
A	scalar
$\underline{a}$	column vector
$\underline{A}$	matrix
$\underline{a}'$	row vector or transpose of $\underline{a}$
$\underline{A}'$	transpose of $\underline{A}$
a^*	value of a at an optimum
$\underline{a}^*$	value of $\underline{a}$ at an optimum
$\underline{A}^*$	value of $\underline{A}$ at an optimum
$\underline{\nabla}$	vector of first partial derivative operators
$\underline{\nabla}^2$	matrix of second partial derivative derivative operators
$\frac{\partial}{\partial x_i}$	ith component of $\underline{\nabla}$
$\frac{\partial^2}{\partial x_i \partial x_j}$	i,jth component of $\underline{\nabla}^2$

LIST OF SYMBOLS USED THROUGHOUT THE TEXT

$\underline{t}$	vector of design variables
$F(\underline{t})$	objective function
$f_i(\underline{t})$	constraint function
P	number of design variables
$\underline{W}$	vector of weight coefficients
σ_{qs}	stress in member s for load case q
$\sigma_{min\ qs}$	minimum permitted stress in s for q
$\sigma_{max\ qs}$	maximum permitted stress in s for q
$t_{min\ j}$	minimum permitted value of design variable j
$t_{max\ j}$	maximum permitted value of design variable j
L	number of load cases
M	number of members
$\underline{\nabla}F(\underline{t})$	vector of first partial derivatives of the objective function
$\underline{\nabla}f_i(\underline{t})$	vector of first partial derivatives of constraint function i
$\underline{d}$	search direction vector
$\emptyset(..)$	objective function for SUMT

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

The author would like to express his appreciation to Dr. W. C. Carpenter, Joyce E. Smith, Ms. E. V. White and Marjorie McGrath Higgins for their help in the preparation of this thesis.

The author also acknowledges the continued assistance of Professors G. Higginson and W. A. Hamilton, the Universities of Durham, England and of Maine, U.S.A. and the financial support of the Science Research Council and of his wife, Joyce.

CHAPTER 1

STATEMENT OF THE PROBLEM

The engineering design problem is to find the optimum, either the maximum or the minimum, of a function of one or more design variables subject to equality and inequality constraints. Examples of engineering design variables are heights, lengths or thicknesses and examples of the function to be optimized, called the objective or merit function, are mass, weight, cost or efficiency. The design is subject to constraints, for example, upper and lower bounds on stresses and deformations, called behavioural constraints, and upper and lower bounds on the design variables, called side constraints. The engineering problem can be stated mathematically as

minimize (or maximize) $F(\underline{t})$ ...1.1

subject to $f_i(\underline{t}) \geqslant 0$, $i = 1,.., R$.

where $\underline{t}$ is a P-vector of design variables t_j, $j = 1,..,P$; $F(\underline{t})$ is the objective function, and $f_i(\underline{t}) \geqslant 0$ are the constraints.

Mathematical Programming methods find the optimum of a function of several variables subject to equality and inequality constraints and can be used on the engineering design problem. Wasiutynski and Brandt[1] in 1963 reviewed the use of classical and contemporary techniques of Mathematical Programming in optimum structural design. Since the early sixties, Sheu and Prager[2] in 1968 and Schmit[3] in 1969 have shown how electronic computation has allowed Mathematical

Programming methods to be used increasingly on structural optimisation problems.

There now exist many suitable Mathematical Programming algorithms, but they vary in the type of problem which they can solve, in the computational effort they require and in their effectiveness at producing an optimal solution. It is desirable, therefore, to predict which methods would be the most appropriate to a particular problem or to a class of problems. The following work makes a comparison of commonly-used algorithms applied to a class of structural optimisation problems. Important considerations in the comparison of the methods are:

1. restrictions of applicability:

   Typical restrictions on the type of problem a method could solve would be requirements for linearity and convexity of the objective or constraint functions.

2. effectiveness:

   The effectiveness required of a method depends on the accuracy required in the solution.

3. computational efficiency:

   The computational efficiency of a method can be measured by the amount of computer time and storage space required to solve the problem.

4. requirements for supplementary programs:

   The additional facilities required by a method could be the evaluation of first or second partial derivatives of the functions, the solutions of sets of linear equations, of linear programming problems, and of one-dimensional search problems.

5. effort for program development:

   The effort for program development depends on the complexity of the method and of the supplementary programs required.

6. feasibility of intermediate solutions:

   For some problems it may be difficult to construct a feasible solution from an infeasible one, feasible intermediate solutions are desirable, though not essential, in case of premature termination of the optimisation process.

The above criteria are used in chapter 2 to select methods to be quantitatively compared in later chapters.

The class of problem considered is the minimization of weight of certain structures subject to stress and design variable limits. The structures considered are pin-jointed plane trusses and plane stress plates. The design variables are, for the trusses, the bar cross-sectional areas and, for the plates, the thicknesses at nodal points of the triangular finite element idealisation. Upper and lower bounds are placed on the design variables and on the stresses in the structural members. The stress is taken as the axial stress in each member for the truss problems and as the effective stress in each constant stress finite element for the plate problems. The optimisation problem for both types of structures can be stated mathematically as:

$$\text{minimize} \quad \underline{w}' \, \underline{t} \qquad \ldots 1.2$$

$$\text{subject to} \quad \sigma_{\min qs} \leq \sigma_{qs} \leq \sigma_{\max qs}, \quad q=1,..,L, s=1,..,M,$$

$$t_{\min j} \leq t_j \leq t_{\max j}, \quad j=1,..,P,$$

where

L is the number of load cases,

M is the number of members,

P is the number of design variables,

$\underline{w}$ is a P-vector of weight coefficients,

$\underline{t}$ is a P-vector of design variables,

$\sigma_{\min qs}$ is the minimum permitted stress in member s for load case q,

$\sigma_{\max qs}$ is the maximum permitted stress in member s for load case q,

σ_{qs} is the stress in member s for load case q,

$t_{\min j}$ is the minimum permitted value of design variable j,

$t_{\max j}$ is the maximum permitted value of design variable j.

Problem 1.2 can be rearranged into the form of 1.1:

$$\text{minimize} \quad \underline{w}' \, \underline{t} \qquad \ldots 1.3,$$

$$\text{subject to} \, (\sigma_{\max qs} - \sigma_{qs}) \geq 0, (\sigma_{qs} - \sigma_{\min qs}) \geq 0,$$

$$q=1,..,L, \quad s=1,..,M,$$

$$(t_{\max j} - t_j) \geq 0, \quad (t_j - t_{\min j}) \geq 0,$$

$$j=1,..,P.$$

Problem 1.3, called a Non-Linear Programming (NLP) problem, has a linear objective function subject to non-linear behavioural constraints and linear side constraints.

Chapter 2 considers methods available for the solution of problem 1.3 and selects methods for comparison in later chapters. Chapter 3 gives details of the solution methods selected for comparison. Chapter 4 describes the methods used to evaluate the objective and constraint functions and their derivatives. Chapter 5 estimates the computational effort required by the optimisation, function and derivative algorithms. Chapter 6 presents test structures used to compare the optimisation algorithms and chapter 7 gives the test results. A summary, conclusions, recommendations and ideas for further research are presented in chapter 8. The appendices give FORTRAN IV program listings of the algorithms used in this study.

CHAPTER 2

METHODS OF SOLUTION FOR THE PROBLEM

2.1 Classification of NLP methods.

There are many methods for solving the general NLP problem and most can be included in one of the following categories:

1. linearization methods,
2. feasible direction methods,
3. transformation methods.

This classification is based on those of Jacoby, Kowalik and Pizzo[4] and of Zoutendijk.[5]

Linearization methods solve the NLP problem using a sequence of Linear Programming problems (LP problems) formed from the NLP problem. Thus an iteration consists of two stages:

i. form a linear approximation at the current point, then

ii. solve the linear approximation by LP methods to give a new solution point.

Feasible direction methods search within the feasible region for an optimal solution along a sequence of 'usable feasible' directions By definition, a search along a 'usable feasible' direction will, for minimization problems, reduce the objective function but maintain feasibility. Thus an iteration consists of two stages:

i. form a usable feasible direction,

ii. search along the direction for a new solution point.

Transformation methods solve the NLP problem indirectly by forming a different, but related, NLP problem. The transformations are such that the solution of the transformed problem coincides with that of the original problem. The transformed problem may often, but not always, be solved as a sequence of problems and may be constrained or unconstrained, depending on the transformations used.

2.2 Linearization methods.

Linearization methods linearize the objective and constraint functions at the initial point. The resulting LP problem is solved by an LP algorithm giving a new solution point. Next, the problem is totally or partially relinearized at the new point and the new LP problem is solved. This procedure is continued until the solutions converge to the optimal solution.

A non-linear objective function can be linearized with a truncated Taylor's series about the current point:

$$F(\underline{t}) = F(\underline{\bar{t}}) + (\underline{\nabla} F(\underline{\bar{t}}))'(\underline{t} - \underline{\bar{t}}) \qquad \ldots 2.2.1$$

Similarly, the constraints can be linearized with truncated Taylor's series:

$$f_i(\underline{t}) = f_i(\underline{\bar{t}}) + (\underline{\nabla} f_i(\underline{\bar{t}}))'(\underline{t} - \underline{\bar{t}}) \geq 0, \; i=1,\ldots,R, \qquad \ldots 2.2.2$$

where $\underline{\bar{t}}$ is the design vector at the current point,

$\underline{\nabla} f_i(\underline{\bar{t}})$ is the vector of first partial derivatives of the ith constraint function with respect to the design variables.

If the original constraints form a convex region, the linearized constraints completely enclose the feasible region. If, however, some of the original constraints are non-convex, then the linearized constraints will cut off some of the feasible region in which the optimal solution may lie.[5] Algorithms must be able to prevent non-convex constraints from slowing or stopping convergence to the optimal solution of the original problem.

Cutting Plane methods (Kelley[6] and Cheney and Goldstein[7]) retain most of the original linearizations of the constraints at each intermediate solution. Only the most active convex constraints are relinearized and the new linearizations are added to the set of constraints. Non-convex constraints are relinearized at each iteration with the new linearizations replacing the old linearizations. A full evaluation of first partial derivatives is not required at each iteration since only a subset of the constraints is relinearized. However, as the method proceeds, the increased problem size increases the computational effort required. Ill-conditioning can arise as more linearizations of each constraint are added.

The Method of Approximate Programming, MAP, (Griffith and Stewart[8]), discards all the old linearizations at each iteration and relinearizes the entire constraint set. Full evaluation of first partial derivatives is required at each iteration, but the problem does not increase in size as the method proceeds. MAP does require additional constraints which limit the size of step that can be taken from the current solution to a new solution. These additional constraints are of the form:

$$\left| t_j - \bar{t}_j \right| \leq \delta_k \ , \ j=1,\ldots,P, \qquad \ldots 2.2.3$$

where δ_k is a positive number preventing large changes in the design variables.

For problems with side constraints, the move limit constraints do not add to the number of constraints since for each design variable one of the upper bound constraints (1 side and 1 move limit constraint) and one of the lower bound constraints (as above) will be redundant. The move limit constraints and complete relinearizations are intended to provide convergence for both convex and non-convex problems although this has not been proved[5]. Possible ill-conditioning is not as severe as on the cutting plane method since each constraint is represented by only one linearization. Intermediate solutions may be infeasible.

Advantages of linearization methods are that functions and first partial derivatives are evaluated no more than once per iteration and one-dimensional searches, which require a number of function evaluations, are replaced by efficient LP methods. However, convergence may be slow when the optimum of the NLP problem does not lie at a vertex of the constraint surfaces or when non-convex constraints are present.[5]

Both the cutting plane method and MAP appear to be apposite to the problem. However the cutting plane method requires additional logic to ensure that old linearizations of non-convex constraints are replaced at each iteration. The computational effort to solve the LP problems increases as optimisation proceeds although some effort can be saved since full derivative evaluations may not be required. When MAP is used, the problem does not increase in size but a full first partial derivative evaluation is required. The main difficulty with MAP is the choice of δ_k. On balance, it appears that MAP is likely to be more efficient than the cutting plane method and since fewer difficulties were anticipated, MAP was selected for comparison with other NLP methods.

2.3 Feasible direction methods.

Feasible direction methods explore the feasible region by searching along directions which reduce the objective function while maintaining feasibility. From the initial point a search direction is found. The design is changed along this search direction until either a minimum is found or until a constraint is encountered. At the new solution point a new search direction is determined and the design is changed by moving along it. A search direction through an intermediate solution point must not violate any constraint for small moves nor allow the objective function to increase. Thus, if $\underline{\bar{t}}$ is an intermediate solution point and I_a are the indices of the constraints active at $\underline{\bar{t}}$, then:

$$f_i(\underline{\bar{t}}) \;=\; 0 \;,\quad i \in I_a \;, \qquad \ldots 2.3.1$$

Expanding such constraints about $\underline{\bar{t}}$ using a truncated Taylor's series gives:

$$f_i(\underline{t}) \;=\; f_i(\underline{\bar{t}}) \;+\; (\underline{\nabla} f_i(\underline{\bar{t}}))'(\underline{t} - \underline{\bar{t}}) \qquad \ldots 2.3.2$$

Let $\underline{d}$ be the search direction through $\underline{\bar{t}}$ and α be a positive scalar, then a new design lying along $\underline{d}$ is given by:

$$\underline{t} = \underline{\bar{t}} + \alpha \underline{d} , \qquad ...2.3.3$$

Substituting equations 2.3.3 and 2.3.1 in equation 2.3.2 gives:

$$f_i(\underline{t}) = \alpha (\underline{\nabla} f_i(\underline{\bar{t}}))'\underline{d} , \qquad ...2.3.4$$

Similarly for the objective function:

$$F(\underline{t}) = F(\underline{\bar{t}}) + \alpha (\underline{\nabla F}(\underline{\bar{t}}))'d , \qquad ...2.3.5$$

The new search direction will be acceptable if

$$f_i(\underline{t}) \geq 0 \text{ and } F(\underline{t}) \leq F(\underline{\bar{t}}) , \qquad ...2.3.6$$

or

$$- (\underline{\nabla} f_i(\underline{\bar{t}}))'\underline{d} \leq 0 , i \in I_a , \qquad ...2.3.7$$

$$+ (\underline{\nabla F}(\underline{\bar{t}}))'\underline{d} \leq 0 .$$

Conditions 2.3.7 are the conditions for a new search direction to be 'usable feasible'. Among the algorithms which satisfy conditions 2.3.7 are Rosen's gradient projection method[9], Gellatly's method[10] and Zoutendijk's methods.[11]

In the gradient projection method, the new direction, $\underline{d}$, is taken as the solution of the equality constrained problem:

$$\text{minimize} \quad (\underline{\nabla} f(\underline{\bar{t}}))'\underline{d} \qquad ...2.3.8$$

$$\text{subject to} \quad - (\underline{\nabla} f_i(\underline{\bar{t}}))'\underline{d} = 0 , i \in I_a$$

$$\underline{d}'\underline{d} = 1 .$$

This problem can be solved using Lagrangean techniques. If the constraints are non-linear the direction may leave the feasible region immediately so that a correction procedure must be applied to maintain feasibility.

In Gellatly's method, the new direction, $\underline{d}$, is taken as the solution of the equality constrained problem:

$$(\underline{\nabla}F(\underline{t}))'\underline{d} = 0 \,, \qquad \ldots 2.3.9$$
$$-(\underline{\nabla}f_i(\underline{t}))'\underline{d} = 1 \,, \; i \in I_a$$

First, the design is moved into the feasible region along the new direction. Next, the objective function is reduced by moving the design along the direction of the gradient of the objective function.

In Zoutendijk's method, the new direction, $\underline{d}$, is taken as the solution of the problem:

maximize $\quad y$ $\qquad$ ...2.3.10

subject to $\quad (\underline{\nabla}F(\underline{t}))'\underline{d} + y \leq 0$,

$\qquad -(\underline{\nabla}f_i(\underline{t}))'\underline{d} + c_i y \leq 0$, $i \in I_a$,

$\qquad$ and $\underline{d}$ is normalized,

where c_i are positive coefficients which can be taken as unity for non-linear constraints and as zero for linear constraints. This problem can be formulated as a LP problem by a suitable normalization of $\underline{d}$.

With the exception of the gradient projection method, feasible direction methods are suitable for the general NLP problem. The gradient projection method is designed for linearly constrained problems, although in combination with a transformation method (section 4) it can be adapted to solve the NLP problem. Gellatly's and Zoutendijk's methods are directly applicable to the NLP problem, and hence the gradient projection method will not be considered further in this study.

For structural problems of the type 1.3, it will be shown that the major computational effort in determining a search direction is the computation of first partial derivatives. Thus a useful measure of computational efficiency is the number of searches required for convergence to the optimum. In Gellatly's method, only alternate searches reduce the objective function, whereas in Zoutendijk's methods every search reduces the objective function. It seems likely that Zoutendijk's method will converge more quickly than Gellatly's method. Accordingly, a method based on the method of Zoutendijk was selected for comparison with other NLP methods.

2.4 Transformation methods.

Transformation methods reduce the degree of difficulty of the constrained NLP problem by forming a simpler, but related NLP problem. Depending on the transformation used, the transformed problem may be solved as a sequence of constrained or unconstrained problems. Transformation methods are of two types: interior point methods and exterior point methods. Interior point methods generate a set of feasible intermediate solutions which converge to the solution of the original problem. Because exterior point methods generate a set of infeasible intermediate solutions, they will not be considered for the solution of problem 1.3.

The Sequential Unconstrained Minimization Technique (SUMT) is an interior point method developed by Fiacco and McCormick[12]. For the SUMT, a new objective function is formed by adding to the original objective function a penalty function (a function of the slackness of the constraints) weighted by an arbitrary scalar.

Thus if the original problem is written as:

minimize $F(\underline{t})$ subject to $f_i(\underline{t}) \geq 0$, $i=1,..,R$; ...2.4.1

then the SUMT formulation is:

solve the sequence of problems:

minimize $\phi(\underline{t},\rho) = F(\underline{t}) + \rho_k P(\ f_i(\underline{t})\ ,\ i=1,..,R)$ ...2.4.2

for $k=1,2,...$

where $\phi(...)$ is the objective function,

ρ_k is an arbitrary scalar, with $\rho_{k+1} < \rho_k$, and

$P(...)$ is the penalty function.

There are two difficulties with SUMT: choice of a suitable value for ρ_1 , and choice of a suitable rate of change for ρ_k. These can be overcome by using the 'Q' transformation of Fiacco and McCormick[12]; the formulation is:

solve the sequence of problems:

minimize $Q(\underline{t},k) = 1/(F_{k-1}(\underline{t}) - F(\underline{t})\ ) + P(f_i(\underline{t}),\ i=1,..,R)$ ...2.4.3,

where $Q(\underline{t},k)$ is the objective function for the kth iteration,

$F_{k-1}(\underline{t})$ is the value of $F(\underline{t})$ at the optimum of $Q(\underline{t},k-1)$.

This formulation was not included for comparison with other NLP methods but in chapter 8 is recommended for further research.

The above SUMT transformations do not take advantage of useful properties such as the possible linearity of some of the constraints or of the original objective function. Fiacco and McCormick[12] suggest that the linear constraints are not included in the penalty function.

The modified SUMT problem is:

solve the sequence of problems:

minimize $\emptyset(\underline{t},\rho) = F(\underline{t}) + \rho_k P(f_i(\underline{t}) \ , \ i \in I_1)$

subject to $f_i(\underline{t}) \geq 0 \ , \ i \in I_2 \ ,$ ...2.4.4

for $k=1,2,\ldots ,$

where I_1 are the indices of the non-linear constraints , and

I_2 are the indices of the linear constraints.

Each $\emptyset(..)$ in problem 2.4.4 can be minimized by a linearization or a feasible direction method.[5] Although the modified SUMT method was not used in this study, it is recommended for further research.

The SUMT formulation of 2.4.2 was chosen as the transformation method to be compared with other NLP methods on problem 1.3. There are two popular penalty functions used with formulation 2.4.2:

1. $$P(\ldots) = \sum_{i=1}^{R} (1/(f_i(\underline{t}))) \ , \qquad \ldots 2.4.5$$

2. $$P(\ldots) = \sum_{i=1}^{R} (-\log(f_i(\underline{t}))) \ , \qquad \ldots 2.4.6$$

Since the evaluation of 'log' requires more computational effort than a division, a penalty function similar to 2.4.5 was selected for use with SUMT. The choice of suitable unconstrained optimisation algorithms for use with SUMT is made in section 5 of this chapter.

2.5 Unconstrained Optimisation Algorithms.

Unconstrained Optimisation Algorithms (UOA) find the values for design variables which optimize an objective function of the variables. Thus, UOAs are suitable for finding, within the feasible region of the original NLP problem, the minima of the transformed objective functions of the SUMT. Among the most efficient UOAs are those which search along a sequence of directions until an optimum is found. Such UOAs have two stages:

i. find a search direction, then

ii. find the optimum along the search direction.

The two stages are repeated until the global optimum is found. An important criterion for choice of one of the UOAs is the computational efficiency of the method. In optimising the problems of the type 1.3, the major computational effort used is that of evaluating the functions and, if required, their derivatives. Thus the computational effort used in optimizing the $\phi(\underline{t},\rho)$ depends upon the number and type of evaluations required to find the search direction (which is dependent on the UOA) and to find the minimum along the search direction (which is independent of the UOA).

UOAs can be categorized by whether they require in the determination of their search directions the evaluation of:

1. functions, their first and second partial derivatives, or
2. functions and their first partial derivatives, or
3. functions only.

It will be shown in a later chapter that derivative evaluations require much more computational effort than function evaluations.

Therefore, derivative methods will be computationally competitive with non-derivative methods only if they require correspondingly fewer one-dimensional searches to find the optimum than the non-derivative methods require.

A number of numerical comparisons of UOAs[13, 14] have shown that among the most efficient methods are those which generate a sequence of conjugate directions or use second derivatives. Accordingly, the following UOAs to be used with SUMT were selected for comparison with other NLP methods:

1. Newton's method with first and second derivatives;[15]
2. Fletcher-Powell's method with first derivatives;[16]
3. Stewart's method with finite difference first derivatives;[17]
4. Powell's method with no derivatives.[18]

2.6 One-dimensional search methods.

Many NLP methods solve the NLP problem by moving the design point through design space along a sequence of search directions until the optimal solution is found. Such methods consist of two stages:

i. determine a search direction - the direction-finding sub-problem, then

ii. determine a move along the search direction - the searching sub-problem.

The searching or one-dimensional search sub-problem finds the move to the boundary of the feasible region and/or the move to the minimum of the objective function. Thus the one-dimensional search problem can be written as:

if $\underline{t} = \underline{\bar{t}} + \alpha\underline{d}$, find the $\underline{t}^* = \underline{\bar{t}} + \alpha^*\underline{d}$ ...2.6.1

such that either

1. $\underline{t}^*$ lies on the boundary of the feasible region, or
2. $\underline{t}^*$ minimizes the objective function,

where $\underline{\bar{t}}$ is the best design point on the previous search,

$\underline{d}$ is the search direction through $\underline{\bar{t}}$ and

α is a scalar specifying the move along $\underline{d}$.

Interval methods or point approximation methods may be used to perform one-dimensional searches. Interval methods find an interval in which the move α^* is known to lie. An interval is chosen. If α^* is not bounded, the interval is expanded. When α^* is bounded, the interval is reduced until the prescribed accuracy is achieved. There are many interval methods but methods based on the Fibonacci numbers or on the Golden Section converge to a prescribed accuracy in the smallest number of iterations.[19]

Point approximation methods estimate the move, α^*, by polynomial approximations. The new point is used in a succeeding approximation for α^*. The process is repeated until successive estimates converge to within the prescribed accuracy. Despite the guaranteed rate of convergence of Fibonacci and Golden Section searches, point approximation methods generally converge more quickly. Powell[18] suggests fitting a second-order polynomial to three function values along the search direction, while Davidon[20] fits a third-order polynomial to two function values and the two corresponding directional derivatives. Davidon's method usually requires fewer approximations than Powell's method. If, however, a derivative evaluation requires much more computational effort than function evaluation, Davidon's method will not be as computationally efficient as Powell's method. A one-dimensional search method based on that of Powell using a second-order polynomial was chosen for use in the solution of the structural problem 1.3.

2.7 Algorithms selected for comparison.

The algorithms selected for comparison in later chapters are:'

1. the Method of Approximate Programming (MAP) - a linearization method,[8]
2. a method based on Zoutendijk's - a method of feasible directions, (MFD)[11],
3. the Sequential Unconstrained Minimization Technique (SUMT) - [12] a transformation method, used in conjunction with:
   i. Newton's method,[15]
   ii. Fletcher-Powell's method,[16]
   iii. Stewart's method, and[17]
   iv. Powell's method.[18]

Of the above methods only Powell's and Stewart's methods do not require the evaluation of explicit first partial derivatives. Newton's method requires the evaluation of second partial derivatives. All the methods except MAP require a one-dimensional search algorithm. MAP and Zoutendijk's method of Feasible directions require a Linear programming algorithm.

The following chapter gives further details of the algorithms and of the modifications required to solve the structural problems 1.2 and 1.3.

CHAPTER 3

DETAILS OF THE ALGORITHMS

3. Introduction.

Chapter 1 introduced the structural problems to be solved and chapter 2 selected methods for solving these problems. This chapter gives details of and modifications to the selected algorithms to handle the structural problems.

The general NLP problem was stated in chapter 1 as:

$$\text{minimize} \quad F(\underline{t}) \qquad \ldots 3.1.1$$

$$\text{subject to} \quad f_i(\underline{t}) \geq 0 \quad , \quad i=1,..,r$$

and the structural problem to be solved was stated as:

$$\text{minimize} \quad \underline{w}'\underline{t} \qquad \ldots 3.1.2$$

$$\text{subject to} \quad \sigma_{min\,qs} \leq \sigma_{qs} \leq \sigma_{max\,qs} \quad , \quad q=1,..,L \; , \; s=1,..,M \; ,$$

$$t_{min\,j} \leq t_j \leq t_{max\,j} \quad , \quad j=1,..,P \; ,$$

or:

$$\text{minimize} \quad \underline{w}'\underline{t} \qquad \ldots 3.1.3$$

$$\text{subject to} \quad 0 \leq (\sigma_{max\,qs} - \sigma_{qs}) \; ,$$

$$0 \leq (\sigma_{qs} - \sigma_{min\,qs}) \; , \quad q=1,..,L \; , \; s=1,..,M \; ,$$

$$0 \leq (t_{max\,j} - t_j) \; ,$$

$$0 \leq (t_j - t_{min\,j}) \; , \quad j=1,..,P \; .$$

3.2 Method of Approximate Programming (MAP).

As described in chapter 2, MAP forms a sequence of linear problems obtained from the NLP problem by linearizing all the non-linear constraints at intermediate solutions. A set of 'move limit' constraints are added to the constraints of the NLP problem to aid stability and convergence of the algorithm. The MAP algorithm can be stated as:

i. select an initial design point;

ii. calculate the first partial derivatives of all the non-linear constraint functions at the current design point;

iii. linearize the objective function and the non-linear constraints;

iv. form the 'move limit' constraints;

v. solve the resulting LP problem using an LP algorithm;

vi. form a new design point from the solution of the LP problem;

vii. terminate if the new and old design points and objective function values converge to within the prescribed accuracy; otherwise go to step ii.

The general LP problem is of the form:

$$\text{minimize} \quad \underline{c}'\,\underline{x} \qquad \ldots 3.2.1$$

$$\text{subject to} \quad \underline{A}\,\underline{x} \leq \underline{b} \quad \text{and} \quad \underline{0} \leq \underline{x}\,,$$

where $\underline{x}$ is the vector of variables,

$\underline{c}$ and $\underline{b}$ are vectors of constants,

$\underline{0}$ is the null vector, and

$\underline{A}$ is the matrix of coefficients.

The objective function of the structural problem 3.1.2 is already linear and does not require linearization for the LP problems. The size of the LP problems can be reduced by combining the linear move limit constraints with the linear side constraints:

If $$\delta_j = \alpha(t_{max\,j} - t_{min\,j}) , \quad 0 < \alpha < 1 , \qquad \ldots 3.2.2$$

is the move limit on the jth design variable,

then the move limit constraints can be written as:

$$\bar{t}_j - \delta_j \leq t_j \leq \bar{t}_j + \delta_j , \quad j=1,..,P , \qquad \ldots 3.2.3$$

where $\bar{t}_j$ is the value of the jth design variable at the current solution point.

The constraints 3.2.3 can be combined with the side constraints of 3.1.2 to give:

$$(t_j^L) = \text{Maximum}(t_{min\,j} , \bar{t}_j - \delta_j) \leq t_j \quad \text{and} \qquad \ldots 3.2.4$$

$$t_j \leq \text{Minimum}(t_{max\,j} , \bar{t}_j + \delta_j) = (t_j^U) , \quad j=1,..,P$$

or

$$(t_j^L) \leq t_j \leq (t_j^U) , \quad j=1,..,P , \qquad \ldots 3.2.5.$$

The total number of constraints in the LP problem can also be reduced by redefining the LP variables thus:

$$tt_j = (t_j - t_j^L) , \quad j=1,..,P , \qquad \ldots 3.2.6$$

Hence constraints 3.2.5 become:

$$0 \leq tt_j \leq (t_j^U - t_j^L) \ , \ j=1,..,P \qquad ...3.2.7$$

The non-linear behavioural constraints in problem 3.1.2 are linearized by expanding in a truncated Taylor's series the constraint functions about the current solution, $\underline{\bar{t}}$:

$$\sigma_{qs} = \bar{\sigma}_{qs} + (\underline{\nabla}\bar{\sigma}_{qs})'(\underline{t} - \underline{\bar{t}}) \qquad ...3.2.8$$

Since $\partial\bar{\sigma}_{qs} / \partial tt_j = \partial\bar{\sigma}_{qs} / \partial t_j$, then

$$\sigma_{qs} = \bar{\sigma}_{qs} + (\underline{\nabla}\bar{\sigma}_{qs})'(\underline{tt} - \underline{\overline{tt}}) \qquad ...3.2.9$$

$$= (\bar{\sigma}_{qs} - (\underline{\nabla}\bar{\sigma}_{qs})'\underline{\overline{tt}}) + (\underline{\nabla}\bar{\sigma}_{qs})'\underline{tt} \qquad ...3.2.10$$

or

$$\sigma_{qs} = \beta_{qs} + (\underline{\nabla}\bar{\sigma}_{qs})'\underline{tt} \qquad ...3.2.11.$$

Equation 3.2.11 substituted into the non-linear constraints of problem 3.1.2 gives the linearized constraints:

$$\sigma_{min\ qs} \leq (\beta_{qs} + (\underline{\nabla}\bar{\sigma}_{qs})'\underline{tt}) \leq \sigma_{max\ qs} \qquad ...3.2.12$$

hence

$$- (\underline{\nabla}\bar{\sigma}_{qs})'\underline{tt} \leq (\beta_{qs} - \sigma_{min\ qs}) \quad \text{and}$$

$$+ (\underline{\nabla}\bar{\sigma}_{qs})'\underline{tt} \leq (\sigma_{max\ qs} - \beta_{qs}) \qquad ...3.2.13$$

which are linear functions of the LP variables, $\underline{tt}$. Rearranging substituting equations 3.2.6 into the objective function of problem 3.1.2 gives

$$\underline{w}'\underline{t} = \underline{w}'\underline{tt} + \underline{w}'\underline{t}^L \qquad ...3.2.14$$

Hence the LP approximation of problem 3.1.2 at $\underline{\bar{t}}$ is:

$$\text{minimize} \quad \underline{w}'\underline{tt} + (\underline{W}'\underline{t}^L) , \qquad \ldots 3.2.15$$

subject to

$$- (\underline{\nabla}\bar{\sigma}_{qs})'\underline{tt} \leq (\beta_{qs} - \sigma_{min\,qs}) ,$$

$$+ (\underline{\nabla}\bar{\sigma}_{qs})'\underline{tt} \leq (\sigma_{max\,qs} - \beta_{qs}) ,$$

$$q=1,..,L \quad , \quad s=1,..,M$$

and

$$tt_j \leq (t_j^U - t_j^L) ,$$

$$0 \leq tt_j ,$$

$$j=1,..,P ,$$

where tt_j , $j=1,..,P$ are the LP variables.

Problem 3.2.15 is of the form 3.2.1 and can be solved by the LP algorithm described later in this chapter. Suitable values for α in 3.2.2 are chosen in chapter 6. The FORTRAN IV program listing of the LP algorithm used in this study is given in the appendices.

3.3 Method of Feasible Directions (MFD).

Feasible direction methods search within the feasible region for an optimal solution along a sequence of usable feasible directions. As described in section 3 of chapter 2 a usable feasible direction will satisfy the following conditions:

$$- (\underline{\nabla}f_i(\underline{t}))'\underline{d} \leq 0 , \; i \in I_a , \qquad \ldots 3.3.1$$

$$+ (\underline{\nabla}F(\underline{t}))'\underline{d} \leq 0 ,$$

where the set I_a are the indices the active constraints.

The algorithm for Zoutendijk's[11] method of feasible directions can be stated as:

i. select an initial feasible design point;

ii. search down the negative of the gradient of the objective function until a minimum of the objective function or a constraint is found;

iii. evaluate the first partial derivatives of the functions;

iv. form the direction finding problem:

$$\begin{aligned} &\text{maximize} \quad y \\ &\text{subject to} \quad (\underline{\nabla}F(\underline{t}))'\underline{d} + y \leq 0 \,, \\ &\qquad -(\underline{\nabla}f_i(\underline{t}))'\underline{d} + c_i y \leq 0 \,, \; i \in I_a \,, \\ &\qquad \underline{d} \text{ is normalized ;} \end{aligned} \qquad \ldots 3.3.2$$

v. solve the direction finding problem;

vi. test the direction for acceptability;

vii. if the direction is acceptable then search along it until a minimum of the objective function or a constraint is found, then go to ix;

viii. if the direction is unacceptable then reduce the number of constraints in the set I_a and go to iv;

ix. terminate if the new and the old design points and objective function values converge to within the prescribed accuracy; otherwise go to iii.

By a suitable normalization of $\underline{d}$, the direction finding problem can be formed as an LP problem. In the direction finding problem, the arbitrary coefficients can be set to unity for the non-linear constraints and to zero for linear constraints. Zoutendijk tests the acceptability of the search direction by examining the value of y. By including in the set I_a all the constraint functions such that

$$0 \leq f_i(\underline{t}) \leq \varepsilon \ , \qquad \text{...3.3.3}$$

and assuming that $c_i = 1$ for the non-linear constraints, then the search direction is usable feasible if:

$$\varepsilon \leq y \qquad \text{...3.3.4}$$

Test 3.3.4 can be obtained by considering equations 2.3.1 to 2.3.7 and the assumption that the search direction is normalized such that $\alpha = 1$ is a meaningful move along the direction. The first order change in $F(\underline{t})$ and $f_i(\underline{t})$ for a unit move along $\underline{d}$ is given by:

$$F(\underline{t}) - F(\underline{\bar{t}}) = (\underline{\nabla} F(\underline{\bar{t}}))'d \qquad \text{...3.3.5}$$

$$f_i(\underline{t}) - f_i(\underline{\bar{t}}) = (\underline{\nabla} f_i(\underline{\bar{t}}))'\underline{d} \qquad \text{...3.3.6}$$

But from 3.3.2:

$$y \leq -(\underline{\nabla} F(\underline{\bar{t}}))'\underline{d} \qquad \text{...3.3.7}$$

and

$$y \leq +(\underline{\nabla} f_i(\underline{\bar{t}}))'\underline{d} \ , \text{ if } c_i = 1 \ , \qquad \text{...3.3.8}$$

Thus

if $\varepsilon \leq y$,

then

$$0 \leq \varepsilon \leq f_i(\underline{t}) - f_i(\underline{\bar{t}}) \qquad \text{...3.3.9}$$

$$0 \leq \varepsilon \leq F(\underline{\bar{t}}) - F(\underline{t}) \qquad \text{...3.3.10}$$

therefore

$$\epsilon \leq f_i(\underline{t}) \qquad \ldots 3.3.11$$

and

$$F(\underline{t}) \leq F(\overline{\underline{t}}) \qquad \ldots 3.3.12.$$

Therefore the direction is usable feasible. If the direction is not acceptable, then ϵ is reduced and the direction finding problem is reformed.

Since the c_i are not dimensionless, the choice of values of unity for the non-linear constraints may not be the most computationally efficient. Furthermore, the test of acceptability 3.3.4 can be incorporated into the direction finding problem. Hence the following formulation of the direction problem was used in this study:

the direction $\underline{d}$ is taken as the solution of the problem

$$\begin{aligned} &\text{maximize} \quad y && \ldots 3.3.13 \\ &\text{subject to} \quad ((\underline{\nabla}F(\overline{\underline{t}}))'\underline{d})/|\Delta F^*| + y \leq 0 &&, \\ &\qquad ((-\underline{\nabla}f_i(\overline{\underline{t}}))'\underline{d})/|\Delta f^*_i| + c_i y \leq -\epsilon/|\Delta f^*_i|, \quad i \in I_a. &&, \end{aligned}$$

and $\underline{d}$ is normalized

where

c_i are dimensionless scalars, = 0 for linear constraints, and > 0 for non-linear constraints,

ΔF^* is the largest possible change in $F(\underline{t})$ for a unit move along any normalized $\underline{d}$ through $\overline{\underline{t}}$ and has units of $F(\underline{t})$, and

Δf_i^* is the largest possible change in $f_i(\underline{t})$ for a unit move along any normalized $\underline{d}$ through $\bar{\underline{t}}$ and has units of $f_i(\underline{t})$.

If $y \leq \varepsilon_m$, where ε_m is a very small positive number, then LP problem 3.3.13 has no feasible region. In this case, ε is reduced and the direction finding problem is reformed.

The values ΔF^* and Δf_i^* depend on the normalization of the search direction, $\underline{d}$. Zoutendijk suggests a number of possible normalizations but some of them require that certain modifications be made to the LP algorithm. The following normalization used in this research does not require modifications to the LP algorithm:

$$\underline{d} \text{ is normalized such that } -D \leq d_j \leq +D\ ,\ j=1,..,P \qquad \text{...3.3.14}$$

Hence the largest possible changes in $F(\underline{t})$ and $f_i(\underline{t})$ for a unit move along any normalized $\underline{d}$ through $\bar{\underline{t}}$ are:

$$\Delta F^* = D\left\|\underline{\nabla} F(\bar{\underline{t}})\right\|_T = D\sum_{j=1}^{P}\left|\underline{\nabla} F(\bar{\underline{t}})\right| \qquad \text{...3.3.15}$$

$$\Delta f_i^* = D\left\|\underline{\nabla} f_i(\bar{\underline{t}})\right\|_T = D\sum_{j=1}^{P}\left|\underline{\nabla} f_i(\bar{\underline{t}})\right| \qquad \text{...3.3.16}$$

where subscript T denotes the 'taxicab' normalization[21] defined by equations 3.3.15 and 3.3.16.

Problem 3.3.13 can be rearranged and combined with normalization 3.2.14 to give:

$$\text{maximize} \quad y \qquad \ldots 3.3.17$$

$$\text{subject to} \quad (\underline{\nabla}F(\underline{t}))'\underline{d} + |\Delta F^*| y \leq 0 ,$$

$$-(\underline{\nabla}f_i(\underline{t}))'d + c_i|\Delta f_i^*| y \leq -\varepsilon , \quad i \in I_a ,$$

$$d_j \leq D ,$$

$$-d_j \leq -D , \quad j=1,..,P .$$

Problem 3.3.17 can be solved by an LP algorithm. The total number of constraints can be reduced by redefining the LP variables thus:

$$dd_j = (d_j + D) , \; j=1,..,P \qquad \ldots 3.3.18$$

hence

$$\text{maximize} \quad y \qquad \ldots 3.3.19$$

$$\text{subject to} \quad (\underline{\nabla}F(\underline{t}))'\underline{dd} + |\Delta F^*| y \leq D \sum_{j=1}^{P} (\underline{\nabla}F(\underline{t}))$$

$$-(\underline{\nabla}f_i(\underline{t}))'\underline{dd} + c_i|\Delta f_i^*| y \leq D \sum_{j=1}^{P} (\underline{\nabla}f_i(\underline{t})) - \varepsilon , \quad i \in I_a$$

$$0 \leq dd_j \leq 2D , \quad j=1,..,P$$

To prevent zig-zagging between a subset of the constraints, Zoutendijk suggested that the set I_a should incorporate the indices of those constraints encountered on some of the previous iterations. Thus in problem 3.3.19, I_a is formed from the union of the two sets I_{act} and I_{rem} which are defined by:

I_{act} = the set of indices for which $0 \leq f_i(\underline{t}) \leq \varepsilon$ ...3.3.20

I_{rem} = the set of indices of the constraints which have been encountered more than once ...3.3.21

hence

$$I_a = (I_{act}) \cup (I_{rem}) \qquad \ldots 3.3.22$$

If the search direction produced in the direction finding problem is rejected, then I_{rem} is emptied and ε is halved or reduced so that at lease one index remains in I_{act}. The set I_a is reformed and a new direction is determined. I_{rem} is updated on succeeding iterations.

To solve the structural problem 3.1.3 by the formulation 3.3.19, the following quantities are required:

$$\underline{\nabla}F(\underline{t}) \text{ and } \underline{\nabla}f_i(\underline{t}) \quad , \; i \in I_a \; .$$

Since

$$F(\underline{t}) = \underline{w}'\underline{t} \quad , \qquad \ldots 3.3.23$$

$$\begin{aligned} f_i(\underline{t}) = \; & (\sigma_{max\,qs} - \sigma_{qs}) \; , \\ & (\sigma_{qs} - \sigma_{min\,qs}) \; , \\ & (t_{max\,j} - t_j) \; , \\ \text{or} \quad & (t_j - t_{min\,j}) \; , \end{aligned}$$

then

$$\underline{\nabla}F(\underline{t}) = \underline{w} \quad ,$$

$$\begin{aligned} \underline{\nabla}f_i(\underline{t}) = \; & -\underline{\nabla}\sigma_{qs} \; , \\ & +\underline{\nabla}\sigma_{qs} \; , \\ & -e_j \; , \\ \text{or} \quad & +e_j \; , \end{aligned} \qquad \ldots 3.3.24$$

where e_j is the jth coordinate direction vector.

The algorithm for the feasible direction method used to solve the structural problem 3.1.3 can be summarized by the following:

i. form an initial feasible design point;

ii. search down the gradient of the objective function until a minimum is found or until a constraint is found;

iii. evaluate first partial derivatives of the functions;

iv. form the direction finding problem 3.3.19 incorporating equations 3.3.15, 3.3.16, 3.3.18, 3.3.20, 3.3.21, 3.3.22, 3.3.23 and 3.3.24;

v. solve the direction finding problem;

vi. if $y \leq \varepsilon_m$, where ε_m is a small positive number, then reduce ε and go to iv;

vii. otherwise, search along the direction for a minimum of the objective function or for a constraint;

viii. terminate if the new and old design points and objective function values converge to within the prescribed accuracy;

ix. otherwise go to iii.

Suitable values for the dimensionless coefficients c_i are selected in chapter 6. A FORTRAN IV program listing of the above algorithm as used is given in the appendices.

3.4 Sequential Unconstrained Minimization Technique (SUMT).

As described in chapter 2, the SUMT is an interior point transformation method. A sequence of unconstrained objective functions (formed from the original objective function and penalty functions) is minimized until the minima converge to within the prescribed accuracy. The SUMT algorithm can be stated as:

i. select an initial feasible design point;

ii. form the transformed objective function $\emptyset(\underline{t},\rho_k)$, k=1;

iii. minimize $\emptyset(\underline{t},\rho_k)$;

iv. terminate if the new design point is satisfactory; otherwise go to v;

v. form the new transformed objective function $\emptyset(\underline{t},\rho_{k+1})$;

vi. estimate the minimum of $\emptyset(\underline{t},\rho_{k+1})$ by extrapolation;

vii. go to iii, with k= k+1;

For the reasons given in chapter 2, the objective functions used in step ii and v are similar to the form:

$$\emptyset(\underline{t},\rho_k) = F(\underline{t}) + \rho_k \left(\sum_{i=1}^{r} \left(1/\left(f_i(\underline{t}) \right) \right) \right) \qquad \ldots 3.4.1.$$

The sequence of values for ρ_k are determined from:

$$\rho_{k+1} = c\,\rho_k \quad , \quad 0 < c < 1 \qquad 3.4.2$$

Equation 3.4.2 requires the values ρ_1 and the coefficient c . The scalar ρ_1 is often determined such that the weighted penalty term is a predetermined proportion of the original objective function at the initial design point:

$$\rho_1 = p\left\{F(\underline{t})/P(f_i(\underline{t}), i=1,\ldots,r)\right\} \qquad \ldots 3.4.3$$

Typical values for p are .01 ,..., .50. However the efficiency and reliability of such an approach is dependent upon the initial design point. If the initial point is close to one or more of the constraints, ρ_1 given by equation 3.4.3 may be too small; alternatively if the initial point is not close to any of the constraints, ρ_1 may be unnecessarily large. Fiacco and McCormick suggest that a 'natural' choice for ρ_1 would be given by the ρ that minimizes the magnitude of the gradient of $\emptyset$ at $\underline{\bar{t}}$, so that $\underline{\bar{t}}$ is close to the minimum of $\emptyset(\underline{t},\rho_1)$. Such a value of ρ_1 could, during the first SUMT iteration, reduce the computational effort used but also reduce the amount by which $\emptyset(\underline{t},\rho_1)$ could be decreased. Nevertheless, the Fiacco and McCormick value for ρ_1 was used in this study and can be obtained from the following:

$$\text{let} \quad (\emptyset = F + \rho_1 P) = (\emptyset(\underline{\bar{t}},\rho_1) = F(\underline{\bar{t}}) + \rho_1 P(f_i(\underline{\bar{t}}), i=1,\dots,r)) \qquad \dots 3.4.4$$

where $\underline{\bar{t}}$ is the current (initial) design,

$$\text{then} \quad \underline{\nabla\emptyset} = \underline{\nabla F} + \rho_1 \underline{\nabla P} \qquad \dots 3.4.5$$

hence ρ_1 is given by the ρ such that $\underline{\nabla\emptyset}'\underline{\nabla\emptyset}$ is a minimum.

But since

$$\underline{\nabla\emptyset}'\,\underline{\nabla\emptyset} = (\underline{\nabla F} + \rho_1 \underline{\nabla P})'(\underline{\nabla F} + \rho_1 \underline{\nabla P}) \qquad \dots 3.4.6$$

then

$$\underline{\nabla\emptyset}'\,\underline{\nabla\emptyset} = \underline{\nabla F}'\underline{\nabla F} + 2\rho_1 (\underline{\nabla F}'\underline{\nabla P}) + \rho_1^2 (\underline{\nabla P}'\underline{\nabla P}) \qquad \dots 3.4.7$$

Differentiating equation 3.4.7 with respect to ρ_1 gives:

$$d(\underline{\nabla\emptyset}'\underline{\nabla\emptyset})/d\rho_1 = 2(\underline{\nabla F}'\underline{\nabla P}) + 2\rho_1(\underline{\nabla P}'\underline{\nabla P}) \qquad \dots 3.4.8.$$

$\underline{\nabla\emptyset}'\underline{\nabla\emptyset}$ has a minimum value when the left hand side of equation 3.4.8 is equal to zero; hence

$$\rho_1 = (-\underline{\nabla F}'\underline{\nabla P})/(\underline{\nabla P}'\underline{\nabla P}) \qquad \dots 3.4.9.$$

The minimum value that $\underline{\nabla\phi}'\underline{\nabla\phi}$ can have is zero; hence from equation 3.4.7:

$$\rho_1 = \frac{(-\underline{\nabla F}'\underline{\nabla P}) \pm \sqrt{(\underline{\nabla F}'\underline{\nabla P})^2 - (\underline{\nabla F}'\underline{\nabla F})(\underline{\nabla P}'\underline{\nabla P})}}{(\underline{\nabla P}'\underline{\nabla P})} \qquad \ldots 3.4.10$$

In this study, the value for ρ_1 was determined using equation 3.4.10. If the quantity under the root sign is negative, then the value for ρ_1 was determined from 3.4.9. If the value for ρ_1 is not positive, for example when $\underline{\nabla F}'\underline{\nabla P} \geq 0$, then ρ_1 was determined from equation 3.4.3.

An efficient choice for the coefficient, c, in equation 3.4.2, is dependent on the accuracy of the search and on the number of unconstrained minimizations attempted. Suitable values for c are determined in chapter 6.

The algorithms used for minimizing the sequence of $\phi(..)$ are detailed in later sections of this chapter.

Preliminary work for this research and other studies[22,23] indicate that computational savings of approximately 30% can be made by incorporating an extrapolation technique into SUMT as in step vi of the algorithm stated above. The technique used in this study is as follows:

i. fit a Lagrangean polynomial through the previous minima ;

ii. predict the minimum of the new objective function using the polynomial ;

iii. search for a minimum along the direction connecting the current design point to the predicted minimum design point ;

iv. proceed with the unconstrained minimization from the new point.

Using a Lagrangean polynomial,[24] the value of a function $y(x)$ can be determined at any value of x as

$$y(x_{n+1}) = \sum_{k=1}^{n} \left(l_k(x_{n+1})\, y(x_k) \right) , \qquad \ldots 3.4.11$$

where

$$l_k(x_{n+1}) = \prod_{\substack{i=1\\ i\neq k}}^{n} (x_{n+1} - x_i) \Big/ \prod_{\substack{i=1\\ i\neq k}}^{n} (x_k - x_i) . \qquad \ldots 3.4.12.$$

Hence the design point at the minimum of the new objective function can be estimated from the following:

let $t_j^*(\rho_{n+1})$ be the estimate the jth design variable at the minimum of the objective function $\phi(\underline{t},\rho_{n+1})$, and

let $t_j^*(\rho_k)$ be the design of the jth design variable at the minimum of the objective functions $\phi(\underline{t},\rho_k)$, then

$$t_j^*(\rho_{n+1}) = \sum_{k=1}^{n} \left(l_k(\rho_{n+1})\, t_j^*(\rho_k) \right) , \qquad \ldots 3.4.13,$$

where

$$l_k(\rho_{n+1}) = \prod_{\substack{i=1\\ i\neq k}}^{n} (\rho_{n+1} - \rho_i) \Big/ \prod_{\substack{i=1\\ i\neq k}}^{n} (\rho_k - \rho_i) \qquad \ldots 3.4.14.$$

But, since $\rho_k = c^{k-1} \rho_1$ $\qquad \ldots 3.4.15,$

then

$$l_k(\rho_{n+1}) = \prod_{\substack{i=1\\ i\neq k}}^{n} \left((c^{n} \rho_1 - c^{i-1} \rho_1)/(c^{k-1} \rho_1 - c^{i-1} \rho_1) \right) \ldots 3.4.16$$

hence

$$l_k(\rho_{n+1}) = \prod_{\substack{i=1\\ i\neq k}}^{n} ((c^{n+1-i} - 1)/(c^{k-i} - 1)) \qquad \ldots 3.4.17$$

The coefficients $l_k(\rho)$ can be determined iteratively by the following recursion formulae developed from equation 3.4.17:

$$l_n(\rho_{n+1}) = (c^n - 1)/(c - 1) \qquad \ldots 3.4.18,$$

$$l_k(\rho_{n+1}) = \frac{(c^n - 1)(c^{n-k} - 1)(l_k(\rho_n))}{(c^{n+1-k} - 1)(c^{k-n} - 1)} \qquad \ldots 3.4.19.$$

The transformed objective functions for SUMT used to solve the structural problem 3.1.3 are given by:

$$\phi(\underline{t},\rho_k) = \underline{w}'\underline{t} + \rho_k (P_1 + P_2) \qquad \ldots 3.4.20$$

where

$$P_1 = (\sigma_{max\,qs} - \sigma_{min\,qs}) \sum_{q=1}^{L} \sum_{s=1}^{M} (1/(\sigma_{max\,qs} - \sigma_{qs}) + 1/(\sigma_{qs} - \sigma_{min\,qs})) \qquad \ldots 3.4.21,$$

$$P_2 = (t_{max\,j} - t_{min\,j}) \sum_{j=1}^{P} (1/(t_{max\,j} - t_j) + 1/(t_j - t_{min\,j})) \qquad \ldots 3.4.22$$

The weighting scalars of equations 3.4.21 and 3.4.22 put the penalty terms in non-dimensional form.

A FORTRAN IV program listing of the SUMT algorithm used in this study is given in the appendices.

3.5 Newton's method[15]

Newton's method can be used with SUMT to minimize the sequence of objective functions $\phi(\underline{t},\rho)$. The method requires the evaluation of functions, first and second partial derivatives. The method used is developed in the following:

Let

$\phi \; = \; \phi(\underline{t},\rho)$,

$\bar{\phi} \; = \; \phi(\bar{\underline{t}},\rho)$ where $\bar{\underline{t}}$ is the current design point,

$\underline{\nabla}\phi \; = \;$ the vector of first partial derivatives of the objective function with respect to the design variables $\underline{t}$,

$\overline{\underline{\nabla}\phi} \; = \; \underline{\nabla}\phi$ at $\bar{\underline{t}}$,

$\underline{\nabla}^2\phi \; = \;$ the matrix of second partial derivatives of the objective function with respect to the design variables,

$\overline{\underline{\nabla}}^2\phi \; = \; \underline{\nabla}^2\phi$ at $\bar{\underline{t}}$,

then expanding ϕ in a truncated Taylor's series about $\bar{\underline{t}}$ gives:

$$\phi = \bar{\phi} + \overline{\underline{\nabla}\phi}^{\,\prime}(\underline{t} - \bar{\underline{t}}) + \tfrac{1}{2}(\underline{t} - \bar{\underline{t}})^{\prime}\,\overline{\underline{\nabla}}^2\phi\,(\underline{t} - \bar{\underline{t}}) \qquad \ldots 3.5.1$$

which has a stationary value when

$$\underline{\nabla}\phi = 0 \qquad \ldots 3.5.2.$$

Differentiating equation 3.5.1 and ignoring higher order terms gives:

$$\underline{\nabla}\phi = \overline{\underline{\nabla}\phi} + \overline{\underline{\nabla}}^2\phi \; (\underline{t} - \bar{\underline{t}}) \qquad \ldots 3.5.3.$$

Hence

$$0 = \overline{\underline{\nabla}\phi} + \overline{\underline{\nabla}}^2\phi \; (\underline{t}^* - \bar{\underline{t}}) \qquad \ldots 3.5.4.$$

Newton's method solves equation 3.5.4 for $\underline{t}^*$ which is an estimate of the design for the minimum of ϕ . When used with SUMT, Newton's method may give a $\underline{t}^*$ which lies in the infeasible region. Newton's method is modified[15] to prevent the design going into the infeasible region, thus:

let $\underline{t}^* = \underline{\bar{t}} + \alpha^* \underline{d}$ ...3.5.5,

where $\underline{d}$ is a search direction,

then $\underline{t}^* - \underline{\bar{t}} = \alpha^* \underline{d}$ ...3.5.6.

Substituting equations 3.5.6 into 3.5.4 gives:

$$0 = \underline{\bar{\nabla}\phi} + \alpha^* \underline{\bar{\nabla}}^2 \phi \; \underline{d} \qquad \ldots 3.5.7.$$

Equation 3.5.7 is solved by setting $\alpha^* = 1$ to yield a search direction $\underline{d}$. Then α^* is determined by searching along $\underline{d}$ for a minimum of ϕ. Thus the algorithm for Newton's method used with SUMT is:

i. calculate $\bar{\phi}$, $\underline{\bar{\nabla}\phi}$ and $\underline{\bar{\nabla}}^2 \phi$; ...3.5.8

ii. solve the set of equations $-\underline{\bar{\nabla}\phi} = \underline{\bar{\nabla}}^2 \phi \; \underline{d}$ for $\underline{d}$;

iii. find the α^* which minimizes ϕ along $\underline{d}$ and replace $\underline{\bar{t}}$ with $\underline{t}^*$, where $\underline{t}^* = \underline{\bar{t}} + \alpha^* \underline{d}$, and go to i.

The process is continued until convergence is achieved to within the prescribed accuracy. The algorithm 3.5.8 will be referred to as Newton(1), hereinafter.

A variation of algorithm 3.5.8 which attempts to reduce the computational effort required will be referred to as Newton(2). Newton(2) omits evaluating $\underline{\nabla}^2 \phi$ on second and subsequent iterations but sets $\underline{\nabla}^2 \phi$ to the values at the initial point.

Newton(1) and Newton(2) as described above were used with SUMT in the tests in chapter 6. A FORTRAN IV program listing of Newton(2) used in this study is given in the appendices.

3.6 Fletcher-Powell's method:[16]

Fletcher-Powell's method can be used to minimize the sequence of objective functions $\phi(\underline{t},\rho)$. This method requires functions and their first partial derivatives and is similar to Newton's method except that the inverse of the Hessian matrix of second partial derivatives is replaced by a matrix which, by improvement after each iteration, converges to the Hessian matrix. The algorithm for Fletcher-Powell's method is:

i. start with an initial design $\underline{t}_0$, and an initial positive definite matrix $\underline{H}_0$, for example, the identity matrix;

ii. calculate $\underline{\nabla\phi}_0$ and set k=0;

iii. determine the search direction $\underline{d}_k$ from the equation

$$\underline{d}_k = - \underline{H}_k \underline{\nabla\phi}_k ;$$

iv. find α^*_k which minimizes ϕ along $\underline{d}_k$ and calculate $\underline{t}_{k+1} = \underline{t}_k + \alpha^*_k \underline{d}_k$;

v. calculate $\underline{\nabla\phi}_{k+1}$ and $\underline{H}_{k+1}$ where

$$\underline{H}_{k+1} = \underline{H}_k + \underline{M}_k + \underline{N}_k ,$$

$$\underline{M}_k = \alpha^*_k (\underline{d}_k \underline{d}'_k)/(\underline{d}'_k \underline{y}_k) ,$$

$$\underline{N}_k = - (\underline{H}_k \underline{y}_k)(\underline{H}_k \underline{y}_k)'/(\underline{y}_k' \underline{H}_k \underline{y}_k) , \text{ and}$$

$$\underline{y}_k = \underline{\nabla\phi}_{k+1} - \underline{\nabla\phi}_k$$

vi. go to iii, with k = k+1.

The process is continued until convergence to within the prescribed accuracy is achieved. Fletcher-Powell's method as described above was used with SUMT in the tests described in chapter 6. A FORTRAN IV program of the method used is given in the appendices.

3.7 Stewart's method[17].

Stewart's method is an extension of Fletcher-Powell's method enabling the use of finite difference first derivatives. In addition to updating the matrix $\underline{H}$, Stewart's method updates the diagonal elements of its inverse $\underline{A}$, which are used in the determination of the finite difference derivatives. Stewart considers the problem of estimating the first derivative of a non-linear function by a linear form and indentifies two major sources of error; - truncation errors and cancellation errors. Truncation errors are caused by the mathematical inadequacy of the derivative approximation. Cancellation errors are caused by the loss of significant figures in finite precision arithmetic. Stewart's method chooses a finite difference step length to that the two sources of error are approximately equal. Stewart shows that this can be done by solving the following equation for each of the coordinate directions:

the step length, δ_j^*, along the jth coordinate direction is given by the solution of

$$|\alpha_{jj}||\delta_j||\Delta\phi_j| - 4|\phi_0||\gamma_j|\eta = 0 \qquad \ldots 3.7.1,$$

where

α_{jj} is the jth diagonal element of the matrix $\underline{A}_k$,

$\Delta\phi_j$ is the change in ϕ for a step δ_j along the jth coordinate direction,

ϕ_0 is the value of the objective function at the current point,

γ_j is the jth component of the last first derivative calculation.

η is an error bound on the function evaluation.

Stewart shows that an approximate solution to equation 3.7.1 is given by either:

$$\delta_j^* = \delta_j\left(1 - (|\alpha_{jj}|\delta_j)/(3|\alpha_{jj}|\delta_j + 4|\gamma_j|)\right) \qquad \ldots 3.7.2$$

for $\gamma_j^2 \geqslant |\alpha_{jj}||\phi_0|\eta$ ...3.7.3

where

$$\delta_j = 2\sqrt{|\phi_0|\eta/|\alpha_{jj}|} \qquad \ldots 3.7.4$$

or

$$\delta_j^* = \delta_j\left(1 - (2|\gamma_j|)/(3|\alpha_{jj}|\delta_j + 4|\gamma_j|)\right) \qquad \ldots 3.7.5$$

for $-\gamma_j^2 \geqslant -|\alpha_{jj}||\phi_0|\eta$ ...3.7.6

where

$$\delta_j = 2\sqrt{|\phi_0||\alpha_j|/\alpha_{jj}^2} \qquad \ldots 3.7.7.$$

Stewart suggests that the value of η should be the larger of (i) the estimate of the error bound on the calculation of ϕ; and (ii) the error bound on the calculation of ϕ by linear expansion about the computer approximation of the current point.

If the step length given by the above equations is greater than some prescribed upper bound, Stewart suggests that a central difference scheme is employed, where δ^*_j is chosen as the positive root of

$$\tfrac{1}{2} \left|\alpha_{jj}\right| \delta^{*2}_j + \left|\gamma_j\right| \delta^*_j - 10^m \left|\phi_0\right| \eta = 0 \qquad \ldots 3.7.8$$

where

10^{-m} is the prescribed upper bound.

The matrix $\underline{A}$ used to find the second derivatives α_{jj} is updated in the following manner:

$$\underline{A}_{k+1} = \underline{A}_k + c_1 \underline{y}_k \underline{y}_k + c_2 (\underline{\nabla\phi}'_k \underline{y}_k + \underline{y}'_k \underline{\nabla\phi}_k) \qquad \ldots 3.7.9,$$

where

$$c_1 = (c_2 / \alpha^*_k - c_2^2 \underline{\nabla\phi}'_k \underline{d}_k) \qquad \ldots 3.7.10, \text{ and}$$

$$c_2 = 1 / \underline{y}_k{}' \underline{d}_k \qquad \ldots 3.7.11.$$

The algorithm for Stewart's method is:

i. start with an initial design, the matrix $\underline{H}_0$ and $\underline{H}_0^{-1} = \underline{A}_0$;

ii. calculate $\underline{\nabla\phi}_0$ and set $k=0$;

iii. determine the search direction $\underline{d}_k$ from $\underline{d}_k = - \underline{H}_k \underline{\nabla\phi}_k$;

iv. find the α^*_k which minimizes ϕ along $\underline{d}_k$ and calculate $\underline{t}_{k+1}$;

v. determine $\eta = \max(\eta_\phi, |\gamma_j t_j \eta_m / \phi_0|)$,
calculate δ_j^* from eqtns. 3.7.2 - 3.7.7, and set $\delta_j^* =$ sign
$$(\alpha_{jj})\ \mathrm{sign}\ (\gamma_j)\delta_j^*\ ;$$

vi. if $\frac{1}{2}|\alpha_{jj}\delta_j^*/\gamma_j| < 10^{-m}$, use a forward first finite difference scheme to obtain $\partial\phi/\partial t_j$,

otherwise calculate δ_j^* from equation 3.7.8 and use a central difference scheme to obtain $\partial\phi/\partial t_j$;

vii. hence calculate $\underline{H}_{k+1}$ and $\underline{A}_{k+1}$;

viii. go to iii, with k = k+1.

The process is continued until convergence to within the prescribed accuracy is achieved.

Stewart's method as described above was used with SUMT in the tests described in chapter 6. A Fortran IV program of the method used is given in the appendices.

3.8 Powell's method[18]

Powell's method can be used with SUMT to minimize the sequence of objective functions $\phi(\underline{t}, \rho)$. The method does not require the evaluation of derivatives, but does require modification for use with SUMT.

Powell's algorithm is:

define a set of P linearly independent directions (e.g. the coordinate directions) as $\underline{d}_1, \underline{d}_2, \ldots, \underline{d}_P$; define the initial point as $\underline{t}_0$

and the objective function at $\underline{t}_r$ as $\phi(\underline{t}_r, \rho)$; then

i. for $r=1,..,P$, find α_r to minimize $\phi(\underline{t}_{r-1} + \alpha \underline{d}_r , \rho)$ and define $\underline{t}_r = \underline{t}_{r-1} + \alpha_r \underline{d}_r$;

ii. find the index R and the quantity D_R = maximum (D_r ; $r=1,..,P$), where $D_r = (\phi(\underline{t}_{r-1}, \rho) - \phi(\underline{t}_r, \rho))$;

iii. define $\phi_0 = \phi(\underline{t}_0, \rho)$ and $\phi_P = \phi(\underline{t}_P, \rho)$ then calculate

$\phi_Q = \phi((2\underline{t}_P - \underline{t}_0),)$;

iv. if either $\phi_0 \leq \phi_Q$ and/or

$\frac{1}{2} D_R (\phi_0 - \phi_P)^2 \leq (\phi_0 - 2\phi_P + \phi_Q)(\phi_0 - \phi_P - D_R)^2$

then go to i with $\underline{t}_0$ replaced by $\underline{t}_P$ and with the old set of directions ; $\underline{d}_1$, $\underline{d}_2$, ... , $\underline{d}_P$;

v. if the tests in iv are not met, then define $\underline{d}_{P+1} = \underline{t}_P - \underline{t}_0$, find the α_{P+1} to minimize $\phi((\underline{t}_P + \alpha \underline{d}_{P+1}), \rho)$,

define $\underline{t}_{P+1} = \underline{t}_P + \alpha_{P+1} \underline{d}_{P+1}$,

then go to i with $\underline{t}_0$ replaced by $\underline{t}_{P+1}$ and with the set of directions : $\underline{d}_1$, $\underline{d}_2$, ... , $\underline{d}_{R-1}$, $\underline{d}_{R+1}$, ... , $\underline{d}_P$, $\underline{d}_{P+1}$.

The tests in step iv of Powell's algorithm combine the following three tests:

1. if $(\phi_0 - 2\phi_P + \phi_Q) < 0$, then take step v ; otherwise

2. if $\phi_O \leq \phi_Q$, then the stationary point of $\phi(\underline{t},\rho)$ lies between $\underline{t}_P$ and $\underline{t}_O$, and the old directions should be used ; otherwise

3. let ϕ_S be the stationary point of a quadratic form fitted to ϕ_O , ϕ_P and ϕ_Q , then

   if $\sqrt{(\phi_O - \phi_S)} - \sqrt{(\phi_P - \phi_S)} \leq \sqrt{D_R}$, then take step v ;
   otherwise use the old directions.

The tests of step iv assume that $\phi(\underline{t},\rho)$ is continuous along the search direction $(\underline{t}_P - \underline{t}_O)$ between $\underline{t} = \underline{t}_P$ and $\underline{t} = 2\underline{t}_P - \underline{t}_O$. However, the formulation with SUMT has $\phi(\underline{t},\rho)$ approaching infinity as $\underline{t}$ approaches the boundary of the feasible region. Since Powell's procedure does not guarantee that $\underline{t} = 2\underline{t}_P - \underline{t}_O$ is in the feasible region, the tests in step iv may not be applicable. A satisfactory test, based on Powell's rationale, to determine whether the new direction should be accepted, can be developed in terms of ϕ_O, ϕ_P and $\phi_M = \phi(\underline{t}_M, \rho)$, where $\underline{t}_M = \frac{1}{2}(\underline{t}_P + \underline{t}_O)$. Assuming that $\phi(\underline{t},\rho)$ is convex, then $\underline{t}_M$ must be in the feasible region.

The three tests combined in step iv can be replaced by the following tests:

1. if $(\phi_O - 2\phi_M + \phi_P) < 0$, then take step v ; otherwise

2. if $(-\phi_O + 4\phi_M - 3\phi_P) < 0$, then the stationary point of $\phi(\underline{t},\rho)$ lies between $\underline{t}_P$ and $\underline{t}_O$, hence the old directions should be used ;

otherwise

3. let ϕ_S be the stationary point of a quadratic form fitted to

   ϕ_O, ϕ_P and ϕ_M, then

   if $\sqrt{(\phi_O - \phi_S)} - \sqrt{(\phi_P - \phi_S)} \leq \sqrt{D_R}$, then take step v,

   otherwise use the old directions.

The above three tests can be combined. Thus steps iii and iv become:

iii. define $\phi_O = \phi(\underline{t}_O, \rho)$ and $\phi_P = \phi(\underline{t}_P, \rho)$, then calculate

$\phi_M = \phi(\tfrac{1}{2}(\underline{t}_P + \underline{t}_O)), \rho)$ and $\phi_S = \phi_M - (\phi_O - \phi_P)^2/(8(\phi_O - 2\phi_M + \phi_P))$;

iv. if $(\phi_O - 2\phi_M + \phi_P) > 0$ and

either (a) $(\phi_O - 4\phi_M + 3\phi_P) > 0$

or (b) $(\phi_O - 4\phi_M + 3\phi_P) < 0$ and

$$\sqrt{(\phi_O - \phi_S)} - \sqrt{(\phi_P - \phi_S)} \geq \sqrt{D_R}$$

then go to i with $\underline{t}_O$ replaced by $\underline{t}_P$ and with the old set of directions : $\underline{d}_1, \underline{d}_2, \ldots, \underline{d}_P$;

With the above modification Powell's method was used with SUMT in the tests described in chapter 6. A Fortran IV program of the method used is given in the appendices.

3.9 One-dimensional search for the minimum of $\emptyset$.

The one-dimensional search algorithm to find the local minimum of the objective function was used in this study in conjunction with the UOAs and SUMT. The algorithm (the programmed with the name ONED) finds a sequence of feasible points, fits a quadratic polynomial to the points, and locates the minimum of the polynomial. One of the previous points is discarded and another polynomial is fitted to the remaining points and the new point. This process is continued until successive estimates of the minimum converge to within the prescribed accuracy. The algorithm can be stated as:

i. set $\alpha_1 = 0$, $\underline{t}_1 = \underline{\bar{t}}$ and $\emptyset_1 = \emptyset(\underline{\bar{t}})$;
determine the largest negative move (amin) and the largest positive move (amax) along $\underline{d}$ that can be taken without violating the linear constraints ; determine the resolution (the minimum distance between two points along $\underline{d}$ that are considered as different points); for derivative methods , form the directional derivative , dy = $\underline{\nabla\emptyset}'\underline{d}$;

ii. form α_2 , the move along $\underline{d}$ to the second point;
for derivative methods : $\alpha_2 = (\emptyset_E - \emptyset_1)/dy$, where $\emptyset_E$ is an estimate of the minimum value of $\emptyset$ along $\underline{d}$,
for non-derivative methods : $\alpha_2 = 5 \times (\text{resolution})$;

iii. if $\alpha_2 \geq$ amax then $\alpha_2 := (\alpha_1 + \text{amax})/3$;
if $\alpha_2 \leq$ amin then $\alpha_2 := (\alpha_1 + \text{amin})/3$;
evaluate $\emptyset_2$ at $\underline{t}_2 = \underline{\bar{t}} + \alpha_2 \underline{d}$;

if any of the non-linear constraints are violated , then

if $\alpha_2 > 0$, set amax = α_2 and go to iii, or

if $\alpha_2 < 0$, set amin = α_2 and go to iii ;

if none of the non-linear constraints are violated, and if the interval of uncertainty (amax-amin) is less than twice the resolution , then terminate at the point with the least value of $\emptyset$;

iv. form α_3 , the move along $\underline{d}$ to the third point; for derivative methods fit a quadratic to $\emptyset$ using dy and two points $\emptyset_1$, α_1 and $\emptyset_2$, α_2 ;

if the quadratic would predict a maximum , find α_3 by extrapolation; for non-derivative methods, find α_3 by extrapolation so that the interval spanned by the three points is three times the interval spanned by the first two points;

v. if $\alpha_3 \geq$ amax, then $\alpha_3 := (\alpha_1 + \alpha_2 + \text{amax})/3$,

if $\alpha_3 \leq$ amin, then $\alpha_3 := (\alpha_1 + \alpha_2 + \text{amin})/3$;

evaluate $\emptyset_3$ at $\underline{t}_3 = \underline{\bar{t}} + \alpha_3 \underline{d}$;

if any of the non-linear constraints are violated , then

if $\alpha_3 > 0$, then set amax = α_3 and go to v , or

if $\alpha_3 < 0$, then set amin = α_3 and go to v ;

reset amax and/or amin if the function values bound the minimum of $\emptyset$ either above and/or below, and if the interval of uncertainty can be reduced; if the interval of uncertainty < 2(resolution), terminate at the point with the least value of $\emptyset$; if the estimate of the second derivative is negative, terminate the search; if it is less than the test value, then discard one of the points and go to iv ;

vi. form α_4, the move along $\underline{d}$ to the fourth point, by fitting a quadratic polynomial to three points using $\phi_1, \alpha_1, \phi_2, \alpha_2, \phi_3$ and α_3;

vii. if $\alpha_4 \geq$ amax, then $\alpha_4 := (\alpha_2 + \alpha_3 + \text{amax})/3$,

if $\alpha_4 \leq$ amin, then $\alpha_4 := (\alpha_1 + \alpha_2 + \text{amin})/3$;

evaluate ϕ_4 at $\underline{t}_4 = \underline{\bar{t}} + \alpha_4 \underline{d}$;

if any of the non-linear constraints are violated, then

if $\alpha_4 > 0$, then set amax $= \alpha_4$ and go to vii, or

if $\alpha_4 < 0$, then set amin $= \alpha_4$ and go to vii;

reset amax and/or amin if the function values bound the minimum of ϕ either above and/or below, and if the interval of uncertainty can be reduced; discard one of the four points;

viii. if the interval of uncertainty is less than twice the resolution, then terminate at the point with the least value of ϕ;

if the remaining three points do not bound the minimum of ϕ, then go to vi;

if the maximum permitted number of quadratic fits has been exceeded then terminate at the point with the least value of ϕ;

go to vi;

A quadratic polynomial is of the form:

$$\emptyset = c_1 \alpha^2 + c_2 \alpha + c_3 \qquad \ldots 3.9.1,$$

where c_1, c_2 and c_3 are coefficients.

Differentiating equation 3.9.1 gives:

$$d\emptyset \,/\, d\alpha = 2c_1 \alpha + c_2 \qquad \ldots 3.9.2.$$

$\emptyset$ has a stationary value, $\emptyset^*$, when $d\emptyset/d\alpha = 0$, or when

$$\alpha = \alpha^* = -c_2 \,/\, 2c_1 \qquad \ldots 3.9.3.$$

Equation 3.9.3 is used in step iv to find α_3 and in step vi to find α_4. In step iv, c_1 and c_2 are determined from the solution of the following three equations:

$$\emptyset_1 = c_1 \alpha_1^2 + c_2 \alpha_1 + c_3 \,, \qquad \ldots 3.9.4,$$

$$\emptyset_2 = c_1 \alpha_2^2 + c_2 \alpha_2 + c_3 \,, \qquad \ldots 3.9.5,$$

$$dy = 2c_1 \alpha_1 + c_2 \,, \qquad \ldots 3.9.6,$$

which yield:

$$c_1 = dy/(\alpha_1 - \alpha_2) - (\emptyset_1 - \emptyset_2)/(\alpha_1 - \alpha_2)^2 \qquad \ldots 3.9.7$$

$$c_2 = dy - 2c_1 \alpha_1 \qquad \ldots 3.9.8.$$

In step vi, c_1 and c_2 are determined from the solution of the equations 3.9.4, 3.9.5 and the following:

$$\emptyset_3 = c_1 \alpha_3^2 + c_2 \alpha_3 + c_3 \qquad \ldots 3.9.9,$$

which yield:

$$c_1 = (\phi_1-\phi_2)/(\alpha_1-\alpha_2)(\alpha_1-\alpha_3) - (\phi_2-\phi_3)/(\alpha_2-\alpha_3)(\alpha_1-\alpha_3) \qquad ...3.9.10$$

$$c_2 = (\phi_2-\phi_3)/(\alpha_2-\alpha_3) - c_1(\alpha_2+\alpha_3) \qquad ...3.9.11.$$

Equation 3.9.3 will predict a minimum provided that the second derivative of ϕ with respect to α is positive, or

$$d^2\phi / d\alpha^2 = 2c_1 \geqslant 0 \qquad ...3.9.12$$

or

$$c_1 \geqslant 0 \qquad ...3.9.13$$

A lower bound on c_1 can be obtained from the following:

consider three points α_i, α_j, and α_m, where $\alpha_m = \frac{1}{2}(\alpha_i + \alpha_j)$ along the search direction $\underline{d}$;

let the function value at the three points be ϕ_i, ϕ_j and ϕ_m;

the coefficient c_1 for this case is :

$$c_1 = (\phi_i - 2\phi_m + \phi_j)/(\tfrac{1}{2})(\alpha_i + \alpha_j)^2 \qquad ...3.9.14.$$

When using limited precision arithmetic, ϕ can not be represented exactly; hence

$$(1-\eta_\phi)\phi_m \leqslant \phi_m \leqslant (1+\eta_\phi)\phi_m \qquad ...3.9.15,$$

where η_ϕ is the error bound on ϕ.

The smallest meaningful absolute value c_1 can have occurs when ϕ_m is given by:

$$\phi_m = (1 \pm \eta_\phi)(\tfrac{1}{2})(\phi_i + \phi_j) \qquad ...3.9.16.$$

Substituting equations 3.9.16 into 3.9.14 gives c_t, a test value for c_1:

$$c_t = \pm 2 \eta_\phi (\phi_i + \phi_j) \Big/ (\alpha_i + \alpha_j)^2 \qquad \ldots 3.9.17.$$

The positive value from equation 3.9.17 is used in step iv to test if a maximum would be predicted and in step vi. For step iv, $\alpha_i = \alpha_1$ and $\alpha_j = \alpha_2$.

For step vi, $\alpha_i = \alpha_1$ and $\alpha_j = \alpha_3$.

In step vii, the point to be discarded is chosen in the following manner:

i. if the latest or the previous best point is an end point of the four points, then discard the other end point and go to iv;

ii. if the four points do not definitely bound the minimum, then discard the end point in the interval furthest from the minimum and go to iv;

iii. if the four points do bound the minimum, then discard the end point which bounds the minimum and go to iv;

iv. return to the search algorithm;

A FORTRAN IV program listing of the above search algorithm used in this study with the UOAs and SUMT is given in the appendices.

3.10 One-dimensional search for the boundary of the feasible region.

The one-dimensional search algorithm to find the boundary of the feasible region was used in this study in conjunction with MFD. The algorithm (programmed with the name FSMOVE) finds a sequence of points within the upper bound move to the boundary defined by the linear constraints. A quadratic polynomial is fitted to each of the non-linear constraints and the smallest positive root is determined. One of the previous points is discarded and another set of polynomials is fitted to the remaining points and the new point. This process is continued until it converges to the boundary to within the prescribed accuracy. The algorithm can be stated as:

i. set $\alpha_1 = 0$, $\underline{t}_1 = \bar{\underline{t}}$, $f_{i1} = f_i(\underline{t}_1)$, $i=1,..,R$ where R is the number of non-linear constraints; determine the resolution (see section 9 of this chapter); determine the largest negative move (amin) and the largest positive move (amax) along $\underline{d}$ to reach the linear constraints; form the negative of the directional derivatives of each of the constraint functions:

$dy_i = -(\underline{\nabla} f_i(\bar{\underline{t}}))'\underline{d}$, $i=1,..,R$;

ii. form α_2, the move along $\underline{d}$ to the second point, from

$\alpha_2 = \text{minimum}\ ((f_i(\bar{\underline{t}}) - \tfrac{1}{2}\epsilon)/dy_i ;\ \ i=1,..,r)$;

if $\alpha_2 >$ amax, then set α_2 = amax;

evaluate $f_{i2} = f_i(\underline{t}_2)$ at $\underline{t}_2 = \bar{\underline{t}} + \alpha_2 \underline{d}$;

if any of the non-linear constraints have been violated, then set amax = α_2 and continue:

iii. form α_3, the move along $\underline{d}$ to the third point, which is an estimate of the move to the nearest non-linear constraint and is found from the solutions of $f_i(\underline{t}) = \varepsilon/2$ and of a quadratic polynomial fitted to the values f_{i1}, α_1, f_{i2}, α_2, and dy_i, $i = 1,..,R$;

if $\alpha_3 >$ amax, then set α_3 = amax;

evaluate $f_{i3} = f_i(\underline{t}_3)$ at $\underline{t}_3 = \bar{\underline{t}} + \alpha_3 \underline{d}$;

if any of the non-linear constraints are violated, then set amax = α_3 and continue ;

iv. reset amax and/or amin if the boundary is bounded either above and/or below providing the interval of uncertainty will be reduced ;

v. form α_4, the move along $\underline{d}$ to the fourth point, in a similar manner as in step iii, except that polynomials are fitted to each of the constraint functions using the values

f_{i1}, α_1, f_{i2}, α_2, f_{i3} and α_3 ;

if $\alpha_4 >$ amax, then set $_4$ = amax ;

evaluate $f_{i4} = f_i(\underline{t}_4)$ at $\underline{t}_4 = \bar{\underline{t}} + \alpha_4 \underline{d}$;

if any of the non-linear constraints are violated, then set amax = α_4 and continue ;

vi. if the interval in which the boundary lies is less than the resolution, then terminate at the point in the feasible region; if the maximum number of quadratic fits has been exceeded, then terminate at the feasible point nearest the boundary;

vii. discard one of the points and go to iv;

The quadratic polynomials are of the form 3.9.1, and the coefficients are determined from the formulae 3.9.7, 3.9.8 or 3.9.10 and 3.9.11. To reduce the amount of computer storage and effort required, advantage was taken of the form of the non-linear constraints for the structural problem 1.2:

$$\sigma_{min\,qs} \leq \sigma_{qs} \leq \sigma_{max\,qs}, \quad q=1,..,L, \; s=1,..,M \qquad \text{...3.10.1.}$$

Thus polynomials were only fitted to each of the σ_{qs}, instead of to each of the $f_i(\underline{t})$. The polynomials are of the form:

$$\sigma_{qs} = c_1\alpha^2 + c_2\alpha + c_3 \qquad \text{...3.10.2,}$$

An estimate of the move to boundary is given by the solution of equation 3.10.2 with the following equations:

$$\sigma_{qs} = (\sigma_{min\,qs} + \frac{\varepsilon}{2}) \text{ or } = (\sigma_{max\,qs} - \frac{\varepsilon}{2}) \qquad \text{...3.10.3,}$$

where ε is given in equation 3.3.3. The four possible solutions of equations 3.10.2 and 3.10.3 are given by:

$$\alpha^* = (-c_2 \pm \sqrt{c_2^2 - 4c_1(c_3 - \sigma_{max\,qs} + \tfrac{1}{2}\varepsilon)})/2c_1, \qquad \text{...3.10.4}$$

or

$$\alpha^* = \left(- c_2 \pm \sqrt{c_2^2 - 4c_1 \left(c_3 - \sigma_{min\ qs} - \tfrac{1}{2}\varepsilon \right)} \right) / 2c_1 \,, \qquad \ldots 3.10.5$$

In step vii, the point to be discarded is chosen in the following manner:

i. order the points such that $\alpha_1 < \alpha_2 < \alpha_3 < \alpha_4$;

ii. if the boundary lies between

a. α_1 and α_2 , discard α_4 , unless it is the newest point , in which case discard α_3 ;

b. α_2 and α_3 , discard α_1 , unless it is the newest point, in which case discard α_4 ;

c. α_3 and α_4 , discard α_1 , unless it is the newest point, in which case discard α_2 .

A FORTRAN IV program listing of the above one-dimensional search algorithm used with MFD in this study is given in the appendices.

3.11 Primal-Dual LP algorithm.

The Primal-Dual LP algorithm, programmed with the name PRMDUL and used in this study with MAP and MFD, finds the optimum of the problem:

$$\text{minimize} \quad \underline{c}'\,\underline{x} \qquad \ldots 3.11.1 \,,$$

$$\text{subject to} \quad \underline{A}\,\underline{x} \leq \underline{b} \,, \quad \underline{0} \leq \underline{x} \,.$$

where the values in $\underline{c}$, $\underline{A}$ and in $\underline{b}$ may be either positive or negative.

The inequalities in 3.11.1 may be converted to equations by the addition of the variables, $\underline{s}$, called slack variables,

$$\text{minimize} \quad \underline{c}'\ \underline{x} + \underline{d}'\ \underline{s} \qquad \ldots 3.11.2\ ,$$

$$\text{subject to} \quad \underline{A}\ \underline{x} + \underline{I}\ \underline{s} = \underline{b}\ ,\ \underline{0} \leq \underline{x}\ ,\ \underline{0} \leq \underline{s}\ .$$

A basic solution may be obtained by setting $\underline{x} = 0$ thus

$$\underline{x} = 0\ ,\ \underline{s} = \underline{b} \qquad \ldots 3.11.3\ .$$

where the variables in $\underline{s}$ are called the basic variables, and in $\underline{x}$ are called the non-basic variables.

The LP algorithm moves from the solution 3.11.3 to the optimum feasible solution by performing elementary row operations on the coefficients of $\underline{c}$, $\underline{d}$, $\underline{A}$, $\underline{I}$ and $\underline{b}$. An optimal solution is found when all the components of the vector $\underline{c}'$ are greater than or equal to zero. The vector $\underline{c}'$ gives the change in the objective function for a unit increase in any of the non-basic variables. A feasible solution is found when all the components of the vector $\underline{b}$ are greater than or equal to zero. The algorithm PRMDUL determines an optimal solution then searches for a feasible optimal solution in the following steps:

i. determine a basic (feasible or infeasible) solution ;

ii. operate on problem 3.11.2 until an optimal (feasible or infeasible) solution is obtained, using the Primal simplex algorithm ;

iii. operate on the optimal solution until a feasible solution is obtained, using the Dual simplex algorithm.

The Primal and Dual LP algorithms operate on the coefficients by selecting the pivot element to give the largest increase in optimality or the largest decrease in infeasibility respectively.

The Primal and the Dual algorithms are well documented[11, 25] and will not be detailed further.

To save computer storage space, a condensed tableau which does not store the matrix $\underline{I}$ but stores the variables associated with the columns of the matrix $\underline{A}$ was used in PRMDUL.

A FORTRAN IV program listing of the algorithm is given in the appendices.

CHAPTER 4

EVALUATION OF FUNCTIONS AND DERIVATIVES

4.1 Functions and their derivatives.

The algorithms described in chapter 3 require some or all of the following quantities:

$$F(\underline{t}) \; , \; f_i(\underline{t}) \; , \; \phi(\underline{t},\rho) \qquad \ldots 4.1.1 \; ,$$

$$\underline{\nabla}F(\underline{t}) \; , \; \underline{\nabla}f_i(\underline{t}) \; , \; \underline{\nabla}\phi(\underline{t},\rho) \qquad \ldots 4.1.2 \; ,$$

and

$$\underline{\nabla}^2 F(\underline{t}) \; , \; \underline{\nabla}^2 f_i(\underline{t}) \; , \; \underline{\nabla}^2 \phi(\underline{t},\rho) \qquad \ldots 4.1.3 \; .$$

The derivatives in equations 4.1.2 and 4.1.3 may be obtained either explicitly by differentiation or by a finite difference technique. Thus, for problem 1.3 ,

$$F(\underline{t}) = \underline{w}' \, \underline{t} \qquad \ldots 4.1.4 \; ;$$

hence, by differentiation,

$$\underline{\nabla}F(\underline{t}) = \underline{w} \; , \text{ and } \underline{\nabla}^2 F(\underline{t}) = \underline{0} \qquad \ldots 4.1.5 \; .$$

Similarly,

$$f_i(\underline{t}) = (\sigma_{max\,qs} - \sigma_{qs}) \; , \; (\sigma_{qs} - \sigma_{min\,qs}) \; , \; (t_{max\,j} - t_j) \text{ or }$$
$$(t_j - t_{min\,j}) \qquad \ldots 4.1.6 \; ;$$

hence

$$\underline{\nabla}f_i(\underline{t}) = \pm \underline{\nabla}\,\sigma_{qs} \; , \text{ or } \pm e_j \; , \text{ respectively} \qquad \ldots 4.1.7 \; ,$$

where e_j is the jth coordinate direction vector ,

and

$$\underline{\nabla}^2 f_i(\underline{t}) = \pm \underline{\nabla}^2 \sigma_{qs} \text{ , or } \underline{0} \text{ , respectively ,} \qquad \ldots 4.1.8 .$$

For the function :

$$\phi(\underline{t},\rho) = F(\underline{t}) + \rho\left(\sum_{i=1}^{r} (1/f_i(\underline{t})) \right) \qquad \ldots 4.1.9 ,$$

differentiation yields :

$$\underline{\nabla}\phi(\underline{t},\rho) = \underline{\nabla}F(\underline{t}) + \rho\left(\sum_{i=1}^{r} (- 1 / f_i(\underline{t})^2)(\underline{\nabla}f_i(\underline{t})) \right) \qquad \ldots 4.1.10 ,$$

and

$$\underline{\nabla}^2 \phi(\underline{t},\rho) = \underline{\nabla}^2 F(\underline{t}) + \rho\left\{\sum_{i=1}^{r}((+(2/f_i(\underline{t})^3) \ \underline{\nabla}f_i(\underline{t}).\underline{\nabla}f_i(\underline{t}))' - (1/f_i(\underline{t})^2) \ \underline{\nabla}^2 f_i(\underline{t}))\right\} \qquad \ldots 4.1.11 .$$

The functions and derivatives in equations 4.1.9, 4.1.10 and 4.1.11 can be obtained from equations 4.1.4 to 4.1.8. Equations 4.1.6 to 4.1.8 require, in particular, the evaluation of

$$\underline{\sigma}, \ \underline{\nabla\sigma} \text{ and } \underline{\nabla}^2 \underline{\sigma} \qquad \ldots 4.1.12 ,$$

for which the algorithms are described in section 2 of this chapter.

An alternative procedure for obtaining derivatives is to use a finite difference derivative scheme. In a forward FD scheme, the ith component of $\underline{\nabla y}$ is given by:

$$\delta y/\delta x_i = (y(\underline{x} + \underline{\delta x}_i) - y(\underline{x}))/\delta x_i \qquad \ldots 4.1.13 ,$$

where

$y(\underline{\bar{x}})$ is the value of $y(\underline{x})$ at $\underline{\bar{x}}$,

$\underline{\delta x}_i$ is the vector $(0,0,..0,\delta x_i ,0,..,0)'$, and

δx_i is a small change in the ith variable , x_i .

Similarly ,

$$\delta^2 y/\delta x_i \delta x_j = \frac{(y(\underline{\bar{x}} + \underline{\delta x}_i + \underline{\delta x}_j) - y(\underline{\bar{x}} + \underline{\delta x}_i) - y(\underline{\bar{x}} + \underline{\delta x}_j) + y(\underline{\bar{x}}))}{(\delta x_i)(\delta x_j)} \quad ...4.1.14 .$$

Hence finite derivatives can be found for the functions $F(\underline{t})$, $f_i(\underline{t})$ and $\emptyset(\underline{t},\rho)$ using equations similar to equations 4.1.13 and 4.1.14.

4.2 Stresses and their derivatives.

The evaluation of derivatives as described in section 1 of this chapter requires some or all of the following quantities :

$$\underline{\sigma} \text{ , } \underline{\nabla}\sigma \text{ and } \underline{\nabla}^2 \sigma \quad ...4.2.1 ,$$

where

$\underline{\sigma}$ is the M x L matrix of member stresses ,

$\underline{\nabla}\sigma$ is the matrix of the first partial derivatives of $\underline{\sigma}$ with respect to the design variables $\underline{t}$, and

$\underline{\nabla}^2\sigma$ is the matrix of the second partial derivatives of $\underline{\sigma}$ with respect to the design variables.

The member stresses, $\underline{\sigma}$, for the truss problems, are taken as the axial stress in each member and for the plate problems, as the effective stress in each constant stress finite element. The effective stress for the plate problems is defined as:

$$\sigma_4 = \sqrt{\sigma_1^2 + \sigma_2^2 - \sigma_1\sigma_2 + 3\sigma_3^2} \qquad \ldots 4.2.2 ,$$

where

σ_4 = the effective stress,

σ_1 = the direct stress in the first coordinate direction ,

σ_2 = the direct stress in the second coordinate direction, and

σ_3 = the shear stress for the first and second coordinate

direction.

In this study, the stiffness matrix method was used to find the quantities in 4.2.1.

The matrix $\underline{\sigma}$ is obtained by solving the matrix equation :

$$\underline{P} = \underline{K}\,\underline{u} \qquad \ldots 4.2.3 ,$$

for $\underline{u}$ and then operating on $\underline{u}$, thus :

$$\underline{\sigma} = \underline{S}\,\underline{u} \qquad \ldots 4.2.4 ,$$

where

$\underline{P}$ is an N x L matrix of N applied nodal loads for L load cases ,

$\underline{u}$ is an N x L matrix of associated deformations ,

$\underline{K}$ is the stiffness matrix , and

$\underline{S}$ is the stress-deformation transformation matrix .

The matrix, $\underline{\nabla\sigma}$, is obtained by differentiating equations 4.2.3 and 4.2.4 with respect to the ith design variable, t_i , to give :

$$\left[\partial \underline{P} / \partial t_i\right] = \left[\left[\partial \underline{K} / \partial t_i\right]\underline{u} + \underline{K}\left[\partial \underline{u} / \partial t_i\right]\right] \qquad \ldots 4.2.5,$$

$$\left[\partial \underline{\sigma} / \partial t_i\right] = \left[\left[\partial \underline{S} / \partial t_i\right]\underline{u} + \underline{S}\left[\partial \underline{u} / \partial t_i\right]\right] \qquad \ldots 4.2.6,$$

where $[.]$ denotes a matrix.

Rearranging equations 4.2.5 and 4.2.6 gives:

$$\left[\left[\partial \underline{P} / \partial t_i\right] - \left[\partial \underline{K} / \partial t_i\right]\underline{u}\right] = \underline{K}\left[\partial \underline{u} / \partial t_i\right] \qquad \ldots 4.2.7,$$

$$\left[\left[\partial \underline{\sigma} / \partial t_i\right] - \left[\partial \underline{S} / \partial t_i\right]\underline{u}\right] = \underline{S}\left[\partial \underline{u} / \partial t_i\right] \qquad \ldots 4.2.8,$$

which are of the same form as equations 4.2.3 and 4.2.4.

The matrix $\underline{\nabla}^2 \sigma$ is obtained by differentiating equations 4.2.7 and 4.2.8 with respect to the jth design variable, t_j, to give:

$$\left[\left[\partial^2 \underline{P} / \partial t_i \partial t_j\right] - \left[\partial^2 \underline{K} / \partial t_i \partial t_j\right]\underline{u} - \left[\partial \underline{K} / \partial t_i\right]\left[\partial \underline{u} / \partial t_j\right]\right] =$$
$$\left[\left[\partial \underline{K} / \partial t_j\right]\left[\partial \underline{u} / \partial t_i\right] + \underline{K}\left[\partial^2 \underline{u} / \partial t_i \partial t_j\right]\right] \qquad \ldots 4.2.9,$$

and

$$\left[\left[\partial^2 \underline{\sigma} / \partial t_i \partial t_j\right] - \left[\partial^2 \underline{S} / \partial t_i \partial t_j\right]\underline{u} - \left[\partial \underline{S} / \partial t_i\right]\left[\partial \underline{u} / \partial t_j\right]\right] =$$
$$\left[\left[\partial \underline{S} / \partial t_j\right]\left[\partial \underline{u} / \partial t_i\right] + \underline{S}\left[\partial^2 \underline{u} / \partial t_i \partial t_j\right]\right] \qquad \ldots 4.2.10.$$

Rearranging equations 4.2.9 and 4.2.10 gives:

$$\left[\left[\partial^2 P/\partial t_i \partial t_j\right] - \left[\partial^2 K/\partial t_i \partial t_j\right]\underline{u} - \left[\partial K/\partial t_i\right]\left[\partial \underline{u}/\partial t_j\right] - \left[\partial \underline{K}/\partial t_j\right]\left[\partial \underline{u}/\partial t_i\right]\right] =$$
$$\underline{K}\left[\partial^2 \underline{u}/\partial t_i \partial t_j\right] \qquad \ldots 4.2.11,$$

and

$$\left[\left\{\partial^2\underline{\sigma}/\partial t_i\partial t_j\right\}-\left[\partial^2\underline{S}/\partial t_i\partial t_j\right]\underline{u}-\left[\partial\underline{S}/\partial t_i\right]\left\{\partial\underline{u}/\partial t_j\right\}-\left[\partial\underline{S}/\partial t_j\right]\left\{\partial\underline{u}/\partial t_i\right\}\right]=\underline{S}\left\{\partial^2\underline{u}/\partial t_i\partial t_j\right\} \qquad \ldots 4.2.12,$$

which are of the same form as equations 4.2.3 and 4.2.4. Equations 4.2.3 and 4.2.4 are solved using the stiffness method. This method can also be used to solve the equations 4.2.7 and 4.2.8 and equations 4.2.11 and 4.2.12, providing the left hand sides of the equations can be formed. Thus the following derivatives are required:

to solve equations 4.2.7 and 4.2.8:

$$\underline{\nabla P}\ ,\ \underline{\nabla K}\ \text{ and }\ \underline{\nabla S} \qquad \ldots 4.2.13$$

and to solve 4.2.11 and 4.2.12:

$$\underline{\nabla}^2\underline{P}\ ,\ \underline{\nabla}^2\underline{K}\ \text{ and }\ \underline{\nabla}^2\underline{S} \qquad \ldots 4.2.14.$$

$\underline{\nabla P}$ is the change in applied forces caused by a change in the design variables. $\underline{\nabla P}$ and $\underline{\nabla}^2\underline{P}$ are null matrices for the structures and the loading under consideration. In the stiffness method, the stiffness matrix, $\underline{K}$, for the assembled structure, can be obtained from the element stiffness matrices, $\underline{k}_j$, for the unassembled structure by using the equation:

$$\underline{K} = \sum_{j=1}^{M} \underline{A}'_j\underline{k}_j\underline{A}_j \qquad \ldots 4.2.15,$$

Where

$\underline{A}_j$ is a displacement transformation matrix which is constant for the structure ;

hence $\underline{\nabla K}$ and $\underline{\nabla}^2 \underline{K}$ can be considered from an elemental level.

The stiffness matrix for element j for the truss problems is given by[26] :

$$\underline{k}_j = \left(\frac{t_j E_j}{l_j} \right) \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad \text{...4.2.16}$$

where

t_j is the cross-sectional area of the jth member,

E_j is Young's modulus of elasticity for the jth member, and

l_j is the length of the jth member.

Hence, differentiating equation 4.2.16 with respect to the ith design variable gives:

$$\left[\partial \underline{k}_j / \partial t_i \right] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \text{ for } i \neq j \qquad \text{...4.2.17}$$

$$\left[\partial \underline{k}_j / \partial t_j \right] = \left(\frac{E_j}{l_j} \right) \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \; i = j \qquad \text{...4.2.18.}$$

Therefore

$$\underline{\nabla}^2 \underline{k}_j = \underline{0} \qquad \text{...4.2.19.}$$

The plane stress plate problems can be analyzed using triangular constant stress finite elements. In this study the design variables were taken as the nodal thicknesses of each element. The stiffness matrix for element s , $\underline{k}_s$, can be shown to be[26] :

$$\underline{k}_s = \frac{E_j (t_{s1} + t_{s2} + t_{s3})}{3 A_{123}} \underline{C} \qquad \ldots 4.2.20 ,$$

where

t_{s1} , t_{s2} , t_{s3} are the three nodal thicknesses for member s ,

A_{123} is the area of the triangular element s, and

$\underline{C}$ is a symmetric matrix of constant coefficients formed from the nodal coordinates and Poisson's ratio.

Differentiating equation 4.2.20 with respect to the ith nodal design variable gives:

$$\partial \underline{k}_s / \partial t_i = \underline{0} \ , \quad i \neq s1,\ i \neq s2 \text{ and } i \neq s3 \qquad \ldots 4.2.21 ,$$

$$\partial \underline{k}_s / \partial t_i = \frac{E_j}{3A_{123}} \underline{C} \ , \quad i=s1,\ i=s2 \text{ or } i=s3 \qquad \ldots 4.2.22 ,$$

Hence $\nabla^2 \underline{k}_s = \underline{0}$, ...4.2.23 .

The matrix $\underline{S}$ transforms nodal displacements into member stresses. For the trusses, the stress transformation matrix for member j is given by:

$$\underline{S}_j = \left(\frac{E_j}{l_j}\right)\left[-1 \quad 1\right] \qquad ...4.2.24\ ;$$

hence

$$\left\{\partial \underline{S}_j / \partial t_i\right\} = \underline{0}\ , \text{ for all } i \text{ and } j \qquad ...4.2.25\ .$$

Thus,

$$\underline{\nabla S} = \underline{0} \text{ and } \underline{\nabla}^2 \underline{S} = \underline{0} \qquad ...4.2.26\ .$$

Similarly, for the plane stress plates,

$$\underline{S}_j = \left(\frac{1}{2A_{123}}\right) \underline{D} \qquad ...4.2.27\ ,$$

where

$\underline{D}$ is a matrix with terms which are functions of the nodal coordinates.

Thus

$$\underline{\nabla S} = \underline{0} \text{ and } \underline{\nabla}^2 \underline{S} = \underline{0} \qquad ...4.2.28\ .$$

Thus for the two types of structure considered, equations 4.2.7 and 4.2.8 simplify to

$$- \left[\partial \underline{K} / \partial t_i\right] \underline{u} = \underline{K}\left\{\partial \underline{u} / \partial t_i\right\} \qquad ...4.2.29\ ,$$

$$+ \left[\partial \sigma / \partial t_i\right] = \underline{S}\left\{\partial \underline{u} / \partial t_i\right\} \qquad ...4.2.30\ .$$

Equations 4.2.11 and 4.2.12 simplify to

$$- \left\{\left[\partial \underline{K}/\partial t_i\right]\left[\partial \underline{u}/\partial t_j\right] + \left[\partial \underline{K}/\partial t_j\right]\left[\partial \underline{u}/\partial t_i\right]\right\} = \underline{K}\left\{\partial^2 \underline{u}/\partial t_i \partial t_j\right\}$$

...4.2.31

and

$$+ \left\{\partial^2 \underline{\sigma} / \partial t_i \partial t_j\right\} = \underline{S}\left\{\partial^2 \underline{u} / \partial t_i \partial t_j\right\} \qquad ...4.2.32 .$$

The solution of equations 4.2.3, 4.2.4, 4.2.29, 4.2.30, 4.2.31 and 4.2.32 gives $\underline{\sigma}$, $\underline{\nabla}\sigma$ and $\underline{\nabla}^2\sigma$ directly for the truss problems, but only gives $\underline{\sigma}_1$, $\underline{\sigma}_2$, $\underline{\sigma}_3$ and their derivatives for the plate problems. The derivatives of the effective stress , $\underline{\sigma}_4$, can be obtained by differentiating equation 4.2.2. Thus, since

$$\sigma_4 = \sqrt{\sigma_1^2 + \sigma_2^2 - \sigma_1\sigma_2 + 3\,\sigma_3^2} = \sqrt{\sigma_4^2} \qquad ...4.2.33,$$

then

$$\left(\frac{\partial\sigma_4}{\partial t_i}\right) = \left(\tfrac{1}{2}\,(\sigma_4^2)^{-\frac{1}{2}}\right)\left(\frac{\partial(\sigma_4^2)}{\partial t_i}\right) \qquad ...4.2.34,$$

hence

$$\left(\frac{\partial\sigma_4}{\partial t_i}\right) = \left(\frac{1}{2\sigma_4}\right)\left\{2\sigma_1\left(\frac{\partial\sigma_1}{\partial t_i}\right) + 2\sigma_2\left(\frac{\partial\sigma_2}{\partial t_i}\right) - \sigma_1\left(\frac{\partial\sigma_2}{\partial t_i}\right) - \sigma_2\left(\frac{\partial\sigma_1}{\partial t_i}\right) + 6\sigma_3\left(\frac{\partial\sigma_3}{\partial t_i}\right)\right\} \qquad ...4.2.35.$$

Differentiating equation 4.2.34 with respect to the jth design variable gives :

$$\left(\frac{\partial^2\sigma_4}{\partial t_i\,\partial t_j}\right) = \left(\frac{-1}{2}\right)\left(\frac{1}{\sigma_4}\right)^2\left(\frac{\partial\sigma_4}{\partial t_j}\right)\left(\frac{\partial(\sigma_4^2)}{\partial t_i}\right) + \left(\frac{1}{2\sigma_4}\right)\left(\frac{\partial^2(\sigma_4^2)}{\partial t_i\,\partial t_j}\right) \qquad ...4.2.36,$$

$$\therefore \left(\frac{\partial^2\sigma_4}{\partial t_i\,\partial t_j}\right) = \left(\frac{-1}{\sigma_4}\right)\left(\frac{\partial\sigma_4}{\partial t_i}\right)\left(\frac{\partial\sigma_4}{\partial t_j}\right) + \left(\frac{1}{2\sigma_4}\right)\left(\frac{\partial^2(\sigma_4^2)}{\partial t_i\,\partial t_j}\right) \qquad ...4.2.37,$$

where

$$\left(\frac{\partial^2(\sigma_4^2)}{\partial t_i\,\partial t_j}\right) = \left\{ 2\left(\frac{\partial\sigma_1}{\partial t_j}\right)\left(\frac{\partial\sigma_1}{\partial t_i}\right) + 2\left(\frac{\partial\sigma_2}{\partial t_j}\right)\left(\frac{\partial\sigma_2}{\partial t_i}\right) - \left(\frac{\partial\sigma_1}{\partial t_j}\right)\left(\frac{\partial\sigma_2}{\partial t_i}\right) - \left(\frac{\partial\sigma_2}{\partial t_j}\right)\left(\frac{\partial\sigma_1}{\partial t_i}\right) + \right.$$

$$6\left(\frac{\partial\sigma_3}{\partial t_j}\right)\left(\frac{\partial\sigma_3}{\partial t_i}\right) + 2\sigma_1\left(\frac{\partial^2\sigma_1}{\partial t_i\,\partial t_j}\right) + 2\sigma_2\left(\frac{\partial^2\sigma_2}{\partial t_i\,\partial t_j}\right) - \sigma_1\left(\frac{\partial^2\sigma_2}{\partial t_i\,\partial t_j}\right) -$$

$$\left. \sigma_2\left(\frac{\partial^2\sigma_1}{\partial t_i\,\partial t_j}\right) + 6\sigma_3\left(\frac{\partial^2\sigma_3}{\partial t_i\,\partial t_j}\right) \right\} \qquad \ldots 4.2.38.$$

Equations 4.2.36 and 4.2.38 can be simplified to give:

$$\left(\frac{\partial^2\sigma_4}{\partial t_i\,\partial t_j}\right) = \left(\frac{1}{2\sigma_4}\right)\left\{ \sigma_1\left(2\left(\frac{\partial^2\sigma_1}{\partial t_i\,\partial t_j}\right) - \left(\frac{\partial^2\sigma_2}{\partial t_i\,\partial t_j}\right)\right) + \left(\frac{\partial\sigma_1}{\partial t_i}\right)\left(2\left(\frac{\partial\sigma_1}{\partial t_j}\right) - \left(\frac{\partial\sigma_2}{\partial t_j}\right)\right) + \right.$$

$$\sigma_2\left(2\left(\frac{\partial^2\sigma_2}{\partial t_i\,\partial t_j}\right) - \left(\frac{\partial^2\sigma_1}{\partial t_i\,\partial t_j}\right)\right) + \left(\frac{\partial\sigma_2}{\partial t_i}\right)\left(2\left(\frac{\partial\sigma_2}{\partial t_j}\right) - \left(\frac{\partial\sigma_1}{\partial t_j}\right)\right) +$$

$$\left. 6\left(\sigma_3\left(\frac{\partial^2\sigma_3}{\partial t_i\,\partial t_j}\right) + \left(\frac{\partial\sigma_3}{\partial t_i}\right)\left(\frac{\partial\sigma_3}{\partial t_j}\right)\right) - 2\left(\frac{\partial\sigma_4}{\partial t_i}\right)\left(\frac{\partial\sigma_4}{\partial t_j}\right) \right\}$$

$$\ldots 4.2.39.$$

Equations 4.2.35 and 4.2.39 enable $\underline{\nabla}\sigma$ and $\underline{\nabla}^2\sigma$ to be evaluated for the plate problems.

The solutions of the stiffness equations, 4.2.3 and 4.2.4, and the derivative equations, 4.2.29 to 4.2.32, are given in the following section of this chapter.

4.3 Solution of the stiffness and derivative equations.

The solution of the stiffness equations, 4.2.3 and 4.2.4, is given by the following algorithm :

i. assemble the basic data for the idealized structure: ...4.3.1 position of nodes, location of members, boundary conditions, material properties and the applied loading ;

ii. determine the band width of the structure stiffness matrix, $\underline{K}$, and the compact storage index matrices ;

iii. calculate the element stiffness constant, stress transformation and weight transformation matrices ;

iv. insert the boundary conditions into the element stiffness constant matrix so that the rigid body degrees of freedom are removed. (This was done by replacing diagonal terms of affected rows and columns with ones and the other terms of affected rows and columns with zeros);

v. determine the design variable values (input data or output data from the optimization algorithms) ; form the element stiffness and structure stiffness matrices ;

vi. decompose the structure stiffness matrix, thus: $\underline{K} = \underline{U}'\underline{U}$, where $\underline{U}$ is an upper triangular banded matrix, using the following formulae :

$$U_{ii} = \sqrt{\left(K_{ii} - \sum_{r=1}^{i-1} (U_{ri} U_{ri})\right)} \qquad \ldots 4.3.2,$$

$$U_{ij} = \left(K_{ij} - \sum_{r=1}^{i-1} (U_{ri} U_{rj})\right)\left(1/U_{ii}\right), \quad j > i \qquad \ldots 4.3.3.$$

vii. solve $\underline{P} = \underline{U}'\underline{U}\,\underline{u}$, using a dummy solution, $\underline{q}$, thus:

$\underline{P} = \underline{U}'\,\underline{q}$, then $\underline{q} = \underline{U}\,\underline{u}$, giving $\underline{u}$, using the following formulae:

$$q_i = \left(P_i - \sum_{r=1}^{i-1} (U_{ri}\, q_r)\right)(1/U_{ii}) \qquad \ldots 4.3.4 \text{ ,}$$

$$u_i = \left(q_i - \sum_{r=i+1}^{A} (U_{ir}\, u_r)\right)(1/U_{ii}) \qquad \ldots 4.3.5 \text{ ,}$$

where

A is the order of the stiffness matrix.

viii. solve $\underline{\sigma} = \underline{S}\,\underline{u}$, giving $\underline{\sigma}$;

ix. for plate problems, determine $\underline{\sigma}_4$ from $\underline{\sigma}_1$, $\underline{\sigma}_2$ and $\underline{\sigma}_3$ using equation 4.2.2 ;

The solution of the first derivative equations 4.2.29 and 4.2.30 is given by the following algorithm, assuming that steps i to ix have been performed already:

for $i = 1,..,P$, ...4.3.6 ,

x. form $\{\partial \underline{K}/\partial t_i\}\underline{u}$;

xi. perform step vii, but with $\{\partial K/\partial t_i\}\underline{u}$ replacing $\underline{P}$, yielding $\{\partial \underline{u}/\partial t_i\}$;

xii. perform step viii, but with $\{\partial \underline{u}/\partial t_i\}$ replacing $\underline{u}$, yielding $\{\partial \underline{\sigma}/\partial t_i\}$;

xiii. for plate problems, determine $\underline{\nabla\sigma}_4$ from $\underline{\nabla\sigma}_1$, $\underline{\nabla\sigma}_2$ and $\underline{\nabla\sigma}_3$ using equation 4.2.35 ;

Steps iii to xii will be repeated for each full partial derivative evaluation. For step x, $\underline{\nabla K}$ has already been formed in step iii.

The solution of the second derivative equations 4.2.31 and 4.2.32 is given by the following algorithm, assuming that steps i to xiii have been performed:

for $i = 1, .., P; j = i, .., P;$ ...4.3.7,

xiv. form $\left\{ \left[\partial \underline{K} / \partial t_i \right] \left\{ \partial \underline{u} / \partial t_j \right\} + \left[\partial \underline{K} / \partial t_j \right] \left\{ \partial \underline{u} / \partial t_i \right\} \right\}$ as in equation 4.2.31 ;

xv. perform step vii, but with the above expression replacing $\underline{P}$;

xvi. perform step viii, but with $\left\{ \partial^2 \underline{u}_2 / \partial t_i \partial t_j \right\}$ replacing $\underline{u}$;

xvii. for plate problems, determine $\underline{\nabla} \underline{\sigma}_4$ using equation 4.2.39 ;

FORTRAN IV program listings of the algorithms used in this study are given in the appendices.

CHAPTER 5

COMPUTATIONAL EFFORT

5.1 Introduction.

Computational efficiency is an important consideration in the comparison of the NLP methods introduced in chapter 1. When using electronic computers, computational efficiency can be measured by the computational effort and the computer storage space required to solve the problem. The storage space required is becoming less important as computers increase in size. However, inefficient data storage and access may increase considerably the computational effort. Recommendations[27] regarding the storage and access of data for the computers used in this study were implemented wherever practical.

The computational effort expended is a function of the efficiency of the computer program for the algorithm. As will be shown later in this chapter and will be seen in the results in chapter 7, the major computational effort is used in the evaluation of functions and their derivatives and since the routines used for these evaluations are common to all the algorithms, any inefficiencies in their programming will affect all the algorithms similarly.

A multiplication with present day computers takes more computer time than an addition or subtraction, and since divisions are relatively few in number, computational effort is often measured by counting the number of multiplications required to perform the operation under consideration. Such an analysis omits the computational effort involved in forming DO loops, array subscript arithmetic, and logical statements, and since multiplications may compose only a small proportion of the computational effort expended, a more realistic measure of computational effort is the amount of computer time required to solve the problem. In this study, Central Processor Unit (CPU) time was used as the measure of computational effort. Included in the CPU time are the times needed to load registers, to execute the instructions, and to store the results. Estimates for CPU time for the IBM 360/67 are developed in the succeeding sections of this chapter using the following procedure :

i. describe the algorithm : see preceding chapters ; ...5.1.1

ii. program the algorithm in FORTRAN : see appendices ;

iii. translate the FORTRAN program into an ASSEMBLER program ;

iv. assign to each of the ASSEMBLER instructions a published average instruction time[28] for the computer used ;

v. sum the times.

It should be noted that step iv was only performed on the instructions which constituted the major computational effort. It should also be noted the times resulting from the application of procedure 5.1.1 are dependent on the programming of the FORTRAN, on the FORTRAN/ASSEMBLER translator (compiler) and on the computer and hence can only be, at best, approximations to the actual effort required.

In an optimisation, the computational effort expended consists of three components :

1. the effort used by the optimisation algorithm ;
2. the effort used to evaluate functions ;
3. the effort used to evaluate derivatives.

Estimates of these three components are considered in the following sections of this chapter.

5.2 Effort used by the optimisation algorithms.

By inspection of the algorithms presented in chapter 3, it can be seen that a large proportion of the computational effort will be expended in performing the following steps in the algorithms:

MAP - step v. solve the LP problem ,

MFD - step v. solve the LP problem to give the search direction, and

SUMT - step iii. minimize $\phi(\underline{t},\rho)$ using the UOAs, where a large proportion of the effort is used in determining the search direction.

In the LP problems, the major computational effort in each LP iteration, is used in finding a pivot element and in transforming the LP tableau. Procedure 5.1.1 was applied to the primal-dual algorithm described in the previous chapter and gave the computational effort to select one pivot and then transform the LP tableau as

$$T_{5.2.1} = 39.3\, r\, c + 144.1\, r + 101.5\, c + 91.9 \qquad \ldots 5.2.1\ ,$$

where

T is the CPU time estimate in microseconds on the IBM 360/67 ,

r is the number of rows in the matrix $\underline{A}$ of the LP problem (3.2.1), and

c is the number of columns in the matrix $\underline{A}$ of the LP problem.

Zoutendijk[11] estimates that the number of iterations required by a primal simplex LP algorithm to produce an optimal solution is between 1 and 2.5 times the number of rows in the primal problem. Similarly the number of iterations required by a dual simplex LP algorithm to produce an optimal solution is between 1 and 2.5 times the number of columns in the primal problem. Observations of preliminary trials on the structural problems indicate that a value of 1.5 times the number of columns gives approximately the number of iterations required by the LP algorithm used. Thus an estimate of the computational effort to find the solution of the LP problems is approximately given by :

$$T_{5.2.2} = 58.95\, c^2 r + 152.3\, c^2 + 216.2\, r\, c + 137.9\, c \qquad \ldots 5.2.2$$

For the class of problems considered, when using MAP,

$$r = P + 2LM\ , \quad c = P \qquad \ldots 5.2.3,$$

and thus r is approximately given by :

$$r = 6P \qquad \ldots 5.2.4\ ;$$

hence the computational effort to find the solution of the LP problems associated with MAP is given by :

$$T_{5.2.5} = 353.7\,P^3 + 1450\,P^2 + 137.9\,P \qquad \ldots 5.2.5.$$

When using MFD for the problems considered, the number of rows is given by:

$$r = P + v2LM \qquad \ldots 5.2.6,$$

where v is the proportion of non-linear constraints considered as active at the current point.

The value of v has been found to give r approximately as :

$$r = 1.5\,P \qquad \ldots 5.2.7\ ;$$

hence the computational effort to find the search direction by solving the LP problems is given by :

$$T_{5.2.8} = 88.43\,P^3 + 476.6\,P^2 + 137.9\,P \qquad \ldots 5.2.8.$$

When using MFD, extra computational effort is used to locate the boundary of the feasible design space. Applying procedure 5.1.1 to the search algorithm (FSMOVE) gives the computational effort necessary to locate one point by using a quadratic fit and associated 'housekeeping' operations as :

$$T_{5.2.9} = 339.0\,R\ , \qquad \ldots 5.2.9\ ,$$

where R is the number of non-linear constraints used.

Assume that $R = 2P$, then

$$T_{5.2.10} = 678.0\,P \qquad \ldots 5.2.10.$$

Typically, only three points are required to locate the boundary, hence

$$T_{5.2.11} = 2034\,P \qquad \ldots 5.2.11.$$

Thus, with MFD, the effort to generate and search along a direction, excluding any function or derivative evaluations, is given approximately by combining equations 5.2.8 with 5.2.11 to give:

$$T_{5.2.12} = 88.4\,P^3 + 476.5\,P^2 + 2172\,P \qquad \ldots 5.2.12$$

When comparing equations 5.2.8 with 5.2.12, it can be seen that the extra effort to search is not as significant as the effort required to generate the direction.

When Newton's method is used with SUMT to minimize $\emptyset(\underline{t},\rho)$, the major computational effort is used in solving equations 3.5.7, which are both linear and symmetric. The procedure 5.1.1 when applied to the equation-solving algorithm (GELS) gave the computational effort to solve equation 3.5.7 as:

$$T_{5.2.13} = 10.68\,P^3 + 112.9\,P^2 + 102.2\,P \qquad \ldots 5.2.13$$

When Fletcher-Powell's method is used with SUMT, the major computational effort is in steps iii and v as described in chapter 3 section 6. The procedure 5.1.1 was applied to those steps in the algorithm (FLEP). The computational effort used to perform steps iii and v is given by:

$$T_{5.2.14} = 129.6\,P^2 + 99.20\,P + 12.61 \qquad \ldots 5.2.14.$$

When Stewart's method is used with SUMT, the computational effort required to perform steps iii and v, as described in chapter 3 section 7 is given as :

$$T_{5.2.15} = 129.6\,P^2 + 400.6\,P + 40.80 \qquad \ldots 5.2.15.$$

When Powell's method is used with SUMT, the computational effort to generate a new search direction and to perform the matrix manipulation prior to each one-dimensional search is given by the following :

$$T_{5.2.16} = 16.21\,P + 56.78 \qquad \ldots 5.2.16.$$

When using a one-dimensional search to find the minimum along a search direction in conjunction with an UOA and SUMT, the computational effort necessary to perform one quadratic fit and associated 'housekeeping' operations, but to exclude any function or derivative evaluations was estimated by procedure 5.1.1 to be:

$$T_{5.2.17} = 23.59\,P + 412.2 \qquad \ldots 5.2.17.$$

A lower bound on the number of new points along the search direction is 3 , although typically between 4 and 9 points along the direction are required to locate the minimum. Thus, assuming that on average, 6½ points are required to locate a minimum and that $T_{5.2.17}$ is approximately equal to the computational effort required to locate any of the points along the search direction, then the computational effort used during a one-dimensional search is given by :

$$T_{5.2.18} = 153.3\,P + 2679 \qquad \ldots 5.2.18.$$

Combining equations 5.2.18 with 5.2.13 to 5.2.16 gives the computational effort to generate and search along a direction; but excludes the effort for any function or derivative evaluations, for Newton's method as :

$$T_{5.2.19} = 10.68\,P^{3} + 112.9\,P^{2} + 255.5\,P + 2679 \qquad \ldots 5.2.19,$$

for Fletcher-Powell's method as :

$$T_{5.2.20} = 129.6\,P^{2} + 252.5\,P + 2692 \qquad \ldots 5.2.20,$$

for Stewart's method as :

$$T_{5.2.21} = 129.6\,P^{2} + 553.9\,P + 2720 \qquad \ldots 5.2.21,$$

and for Powell's method as :

$$T_{5.2.22} = 169.5\,P + 2736 \qquad \ldots 5.2.22.$$

Comparing equations 5.2.13 to 5.2.16 with 5.2.19 to 5.2.22, it can be seen that, with the exception of Powell's method, the computational effort to perform a search is not very significant compared with the effort to generate the direction.

5.3 Effort used in evaluating functions.

The function evaluations required by the optimisation algorithms are the determination of :

$$F(\underline{t})\ ,\ f_i(\underline{t}) \text{ and/or } \emptyset(\underline{t},\rho) \qquad \ldots 5.3.1\ ,$$

in which the major computational effort is used in determining the stresses, $\underline{\sigma}$. The algorithm used to determine $\underline{\sigma}$ is given in section 3 of chapter 4. In the optimisation process, steps i to iv of the algorithm will be performed only once, whereas steps v to viii will be repeated many times. Therefore the computational effort used in steps i to iv will not be considered further.

All the major computational manipulation in steps v to viii can be formed from the following operations :

1. locate an element in a vector, using subscript arithmetic, and post it into another vector ;
2. add the product of an element in another matrix and an element in a vector to an element in a matrix ;
3. add the product of two elements in a matrix to a scalar ;
4. replace an element in a matrix by the difference of the element and a scalar ;
5. replace a diagonal element in a matrix by the reciprocal of its square root ;
6. replace an element of a matrix by its product with an element of a vector.

Applying procedure 5.1.1 gives the following results:

Table 5.3.2: Computational effort for basic operations

Operation	CPU time (microseconds)
1	19.24
2	19.31
3	17.96
4	23.22
5	115.28
6	15.53

Using the values in table 5.3.2, the computational effort to complete step v is given approximately by :

$$T_{5.3.3} = 19.24\ (M)(C) + 19.31\ (M)(E) \qquad \ldots 5.3.3\ ,$$

where M is the number of members ,

C is the number of design variables which affect the member stiffness matrix, and

E is the number of elements in the upper part of the member stiffness matrix.

Similarly, the computational effort to complete step vi is given approximately by :

$$T_{5.3.4} = 17.96\ (B)(B-1)(3A-2B+1)/6 + 23.22\ (B-1)(2A-B)/2 + 115.28\ A + 15.53\ (B-1)(2A-B)/2 \qquad \ldots 5.3.4\ ,$$

or

$$T_{5.3.5} \doteq 17.96\ (B)(B-1)(3A-2B+1)/6 + 38.75\ (B-1)(2A-B)/2 + 115.28A \qquad \ldots 5.3.5\ ,$$

where A is the order of the system stiffness matrix, and

B is the bandwidth of the system stiffness matrix .

The computational effort to complete step vii is given approximately by:

$$T_{5.3.6} = (2)\left((19.31\ (B-1)(2A-B)(L)/2) + (38.75\ (A)(L))\right) \qquad ...5.3.6$$

where L is the number of load cases.

The computational effort to complete step viii is given approximately by :

$$T_{5.3.7} = 19.24\ (L)(M)(D) + 19.31\ (L)(M)(D)(S) \qquad ...5.3.7,$$

where D is the number of nodal displacements associated with each member, and

S is the number of components of stress associated with each member.

Equations 5.3.3 to 5.3.7 can be simplified by substituting values for the variables from structural problems of the type given in chapter 6. Thus, for the truss problems, assume that

$$M = P\ ,\ C = 1,\ E = 10,\ A = 2P + 2,\ B = P + 3,\ L = 2,\ D = 4, \text{ and } S = 1\ , \qquad ...5.3.8.$$

substituting equations 5.3.8 into 5.3.3 gives:

$$T_{5.3.9} = 19.24\ P + 193.1\ P = 212.3\ P \qquad ...5.3.9,$$

into 5.3.5 gives :

$$T_{5.3.10} = 17.96\ (P+3)(P+2)(4P+1)/6 + 38.75\ (P+2)(3P+1)/2 + 115.28\ (2P+2) \qquad ...5.3.10\ ,$$

or

$$T_{5.3.11} = 12.0\ P^3 + 121.0\ P^2 + 453.0\ P + 287.3 \qquad \ldots 5.3.11\ ,$$

into 5.3.6 gives :

$$T_{5.3.12} = (2)\ \Big(\ (\ 19.31\ (P+2)(3P+1)/2) + (38.75(2P+2)(2))\Big) \qquad \ldots 5.3.12$$

or

$$T_{5.3.13} = 57.9\ P^2 + 445.2\ P + 348.6 \qquad \ldots 5.3.13.$$

into 5.3.7 gives :

$$T_{5.3.14} = 19.24\ (2)(P)(4) + 19.31\ (2)(P)(4)(1) \qquad \ldots 5.3.14,$$

or

$$T_{5.3.15} = 308.4\ P \qquad \ldots 5.3.15.$$

Similarly, for the plate problems, assume that

$M = 1.5P-4$, $C = 3$, $E = 21$, $A = 2P$, $B = .25P + 6$, $L = 2$, $D = 6$ and

$S = 4$, ...5.3.16.

Substituting equations 5.3.16 into 5.3.3 gives:

$$T_{5.3.17} = 19.24(1.5P-4)(3) + 19.31(1.5P-4)(21) = 694.9\ P - 1853 \qquad \ldots 5.3.17,$$

into 5.3.5 gives:

$$T_{5.3.18} = 17.96(.25P+6)(.25P+5)(6P-.5P-11)/6 + 38.75(.25P+5)(4P-.25P-6)/2 + 115.28(2P) \qquad \ldots 5.3.18,$$

or

$$T_{5.3.19} = 1.03\ P^3 + 61.4\ P^2 + 968.1\ P - 2073 \qquad \ldots 5.3.19,$$

into 5.3.6 gives :

$$T_{5.3.20} = (2)\ \Big((19.31)(.25P-5)(4P-.25P-6)/2 + (38.75)(2P)(2)\Big) \qquad \ldots 5.3.20,$$

or

$$T_{5.3.21} = 18.1\,P^2 + 488.1\,P - 579.3 \qquad \ldots 5.3.21,$$

into 5.3.7 gives :

$$T_{5.3.22} = 19.24\,(2)(1.5P-4)(6) + 19.31\,(2)(1.5P-4)(6)(4) \qquad \ldots 5.3.22,$$

or

$$T_{5.3.23} = 1737\,P - 4631 \qquad \ldots 5.3.23.$$

From equations 5.3.9, 5.3.11, 5.3.13 and 5.3.15, the computational effort needed to evaluate the stresses in the trusses is given approximately by :

$$T_{5.3.24} = 12.0\,P^3 + 178.9\,P^2 + 1420\,P + 635.9 \qquad \ldots 5.3.24.$$

From equations 5.3.17, 5.3.19, 5.3.21 and 5.3.23, the computational effort needed to evaluate the stresses in the plates is given approximately by :

$$T_{5.3.25} = 1.03\,P^3 + 79.5\,P^2 + 3888\,P - 9136 \qquad \ldots 5.3.25.$$

5.4 Effort used in evaluating derivatives.

The derivative evaluations required by the optimisation algorithms are the determination of :

$$\underline{\nabla}F(\underline{t}),\ \underline{\nabla}f_i(\underline{t}),\ \underline{\nabla}\emptyset(\underline{t},\rho),\ \underline{\nabla}^2 F(\underline{t}),\ \underline{\nabla}^2 f_i(\underline{t}) \text{ and } \underline{\nabla}^2 \emptyset(\underline{t},\rho) \qquad \ldots 5.4.1,$$

in which the major computational effort is used to determine the derivatives of the stresses. The algorithms used in this study are given in section 3 of chapter 4. Using the values in 5.3.2, the computational effort to complete step x P times is given approximately by :

$$T_{5.4.2.} = \left(19.24\ (L)(D)(M)(C)/(P) + 19.31\ (L)(D^2)(M)(C)/(P)\right)(P) \quad \ldots 5.4.2.$$

The effort to complete step xi, P times, is given by:

$$T_{5.4.3} = \left(T_{5.3.6}\right)(P) \quad \ldots 5.4.3$$

and to complete step xii, P times, is given by:

$$T_{5.4.4} = \left(T_{5.3.7}\right)(P) \quad \ldots 5.4.4,$$

Thus, for the truss problems, substituting the values 5.3.8 into equation 5.4.2 gives:

$$T_{5.4.5} = 19.24\ (2)(4)(P)(1) + 19.31\ (2)(16)(P)(1) = 771.8\ P \quad \ldots 5.4.5,$$

into equation 5.4.3 gives:

$$T_{5.4.6} = 57.9\ P^3 + 445.2\ P^2 + 348.6\ P \quad \ldots 5.4.6,$$

and into equation 5.4.4 gives:

$$T_{5.4.7} = 308.4\ P^2 \quad \ldots 5.4.7.$$

Similarly for the plate problems, substituting the values 5.3.16 into equation 5.4.2 gives:

$$T_{5.4.8} = 19.24\ (2)(6)(1.5P-4)(3) + 19.31\ (2)(36)(1.5P-4)(3) = 5210\ P - 16{,}960 \quad \ldots 5.4.8,$$

into equation 5.4.3 gives:

$$T_{5.4.9} = 18.1\ P^3 + 488.1\ P^2 - 579.3\ P \quad \ldots 5.4.9,$$

and into equation 5.4.4 gives:

$$T_{5.4.10} = 1737\,P^2 - 4631\,P \qquad \ldots 5.4.10.$$

From equations 5.4.5 to 5.4.10, the computational effort to evaluate the first derivatives of stress, assuming that the stresses have already been evaluated, is given approximately by:

$$T_{5.4.11} = 57.9\,P^3 + 753.6\,P^2 + 1120\,P \qquad \ldots 5.4.11,$$

for trusses, and approximately by:

$$T_{5.4.12} = 18.1\,P^3 + 2225\,P^2 - 16960 \qquad \ldots 5.4.12,$$

for the plates.

The computational effort to evaluate the first derivatives of stress using finite differences, is given approximately by:

$$T_{5.4.13} = 12.0\,P^4 + 178.9\,P^3 + 1420\,P^2 + 635.9\,P \qquad \ldots 5.4.13,$$

for the trusses, and by:

$$T_{5.4.14} = 1.03\,P^4 + 79.5\,P^3 + 3888\,P^2 - 9136\,P \qquad \ldots 5.4.14,$$

for the plates.

The computational effort to complete step xiv, P(P+1)/2 times, is given by:

$$T_{5.4.15} = \left(T_{5.4.2}\right)(2)(P+1)/2 \qquad \ldots 5.4.15,$$

to complete step xv, P(P+1)/2, times is given by:

$$T_{5.4.16} = \left(T_{5.4.3}\right)(P+1)/2 \qquad \ldots 5.4.16,$$

and to complete step xvi, P(P+1)/2 times, is given by:

$$T_{5.4.17} = \left(T_{5.4.4}\right)^{(P+1)/2} \qquad \ldots 5.4.17.$$

Thus, substituting the values 5.3.8 for the truss problems into equation 5.4.15 gives:

$$T_{5.4.18} = 771.8\ P^2 + 771.8\ P \qquad \ldots 5.4.18,$$

into equation 5.4.16 gives:

$$T_{5.4.19} = 28.95\ P^4 + 251.6\ P^3 + 396.9\ P^2 + 348.6\ P \qquad \ldots 5.4.19,$$

and into equation 5.4.17 gives:

$$T_{5.4.20} = 154.2\ P^3 + 154.2\ P^2 \qquad \ldots 5.4.20.$$

Similarly, for the plate problems, substituting the values 5.3.16 into equation 5.4.15 gives:

$$T_{5.4.21} = 5210\ P^2 - 11{,}750\ P - 16{,}960 \qquad \ldots 5.4.21,$$

into equation 5.4.16 gives:

$$T_{5.4.22} = 9.05\ P^4 + 253.1\ P^3 - 45.6\ P^2 - 289.7\ P \qquad \ldots 5.4.22,$$

and into equation 5.4.17 gives:

$$T_{5.4.23} = 868.5\ P^3 - 1437\ P^2 - 2316 \qquad \ldots 5.4.23.$$

From equations 5.4.18 to 5.4.23, the computational effort to evaluate the second derivatives of stress, assuming that the stresses and their first derivatives have already been evaluated, is given approximately by:

$$T_{5.4.24} = 28.95\ P^4 + 405.8\ P^3 + 1323\ P^2 + 1120\ P \qquad \ldots 5.4.24,$$

for the trusses, and approximately by:

$$T_{5.4.25} = 9.05\,P^4 + 1122\,P^3 + 3728\,P^2 - 12040\,P \quad ...5.4.25,$$

for the plates.

The computational effort to evaluate the second derivatives of stress using finite differences, is given approximately by:

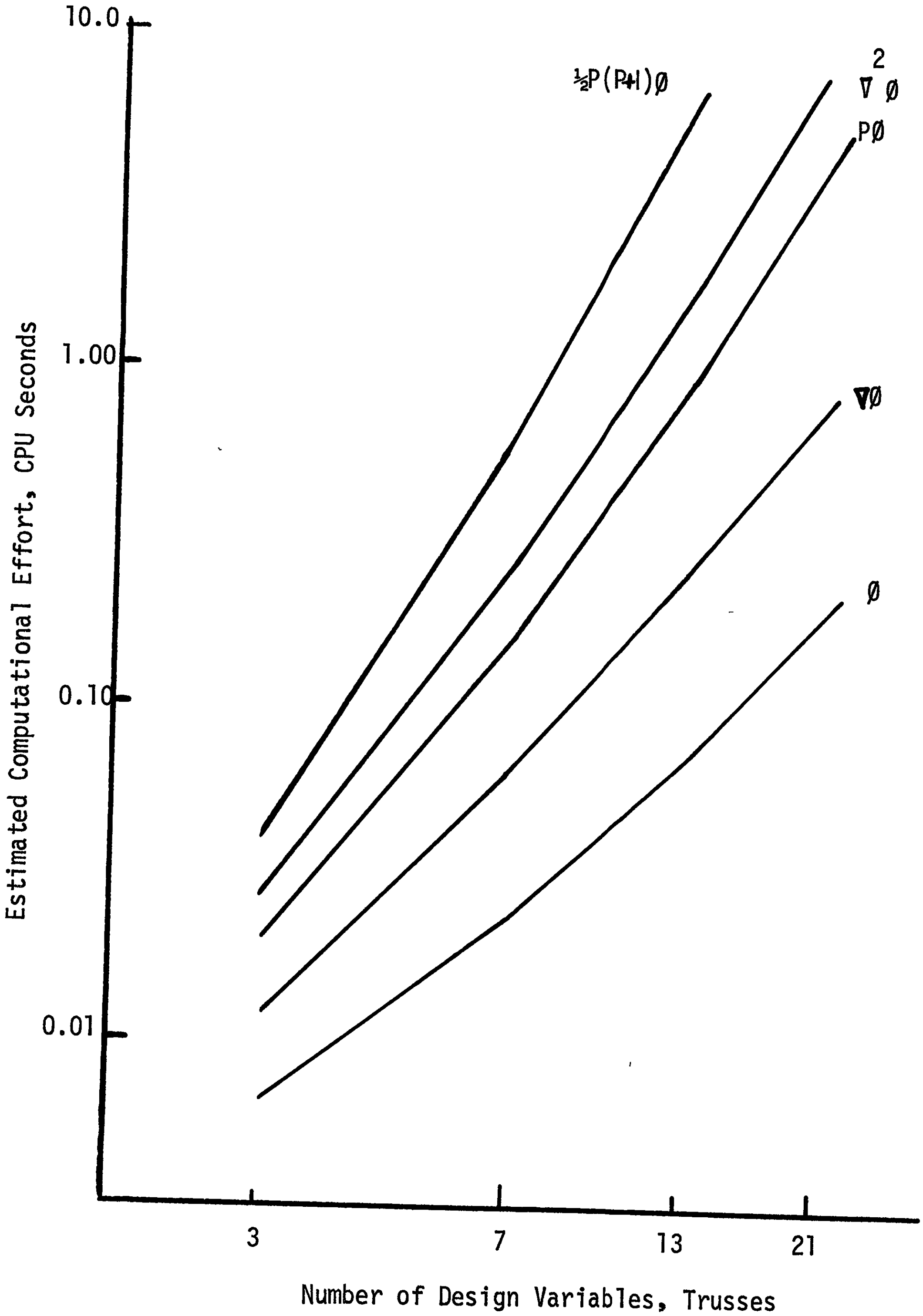
$$T_{5.4.26} = 6.00\,P^5 + 95.5\,P^4 + 798.9\,P^3 + 1027\,P^2 + 317.9\,P \quad ...5.4.26,$$

for the trusses, and approximately by:

$$T_{5.4.27} = .515\,P^5 + 40.3\,P^4 + 1984\,P^3 - 2624\,P^2 - 4568\,P - 4568 \quad ...5.4.27,$$

for the plates.

Figures 5.4.28 and 5.4.29 respectively plot estimated computational effort required for the trusses and plates of chapter 6 using an IBM 360/67 computer to evaluate a function (as given approximately by equations 5.3.24 and 5.3.25), a first derivative (as given by equations 5.4.11 to 5.4.14), and a second derivative (as given by equations 5.4.24 to 5.4.27).



10.0
1.00
0.10
0.01
Estimated Computational Effort, CPU Seconds
½P(P+1)Ø
∇²Ø
PØ
∇Ø
Ø
3
7
13
21
Number of Design Variables, Trusses

FIGURE 5.4.28

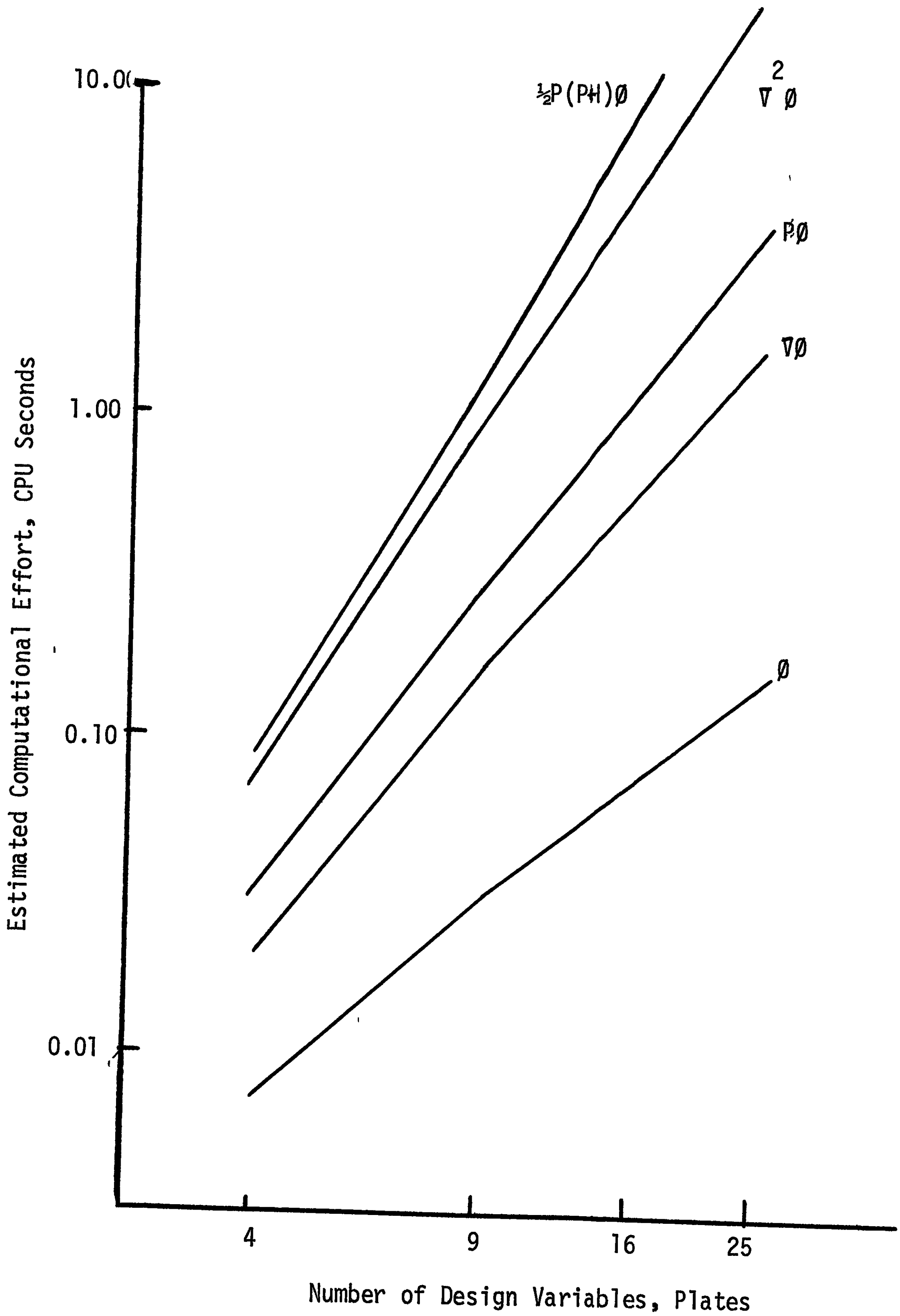

10.00
1.00
0.10
0.01
Estimated Computational Effort, CPU Seconds
½P(PH)Ø
∇^2 Ø
PØ
∇Ø
Ø
4
9
16
25
Number of Design Variables, Plates

FIGURE 5.4.29

Table 5.4.30: Estimated function and derivative effort ratios, $E_{A,B}$.

A	B	P-BAR TRUSSES , P = 3	7	13	21	P-NODE PLATES , P = 4	9	16	25
$\nabla\phi$	ϕ	1.81	2.75	3.56	4.05	2.55	5.34	8.07	10.8
$\nabla^2\phi$	ϕ	4.18	12.0	25.8	45.3	11.0	32.4	76.6	150.
$P\times\phi$	ϕ	3.00	7.00	13.0	21.0	4.00	9.00	16.0	25.0
$\frac{(P^2+P)}{2}\phi$	ϕ	6.00	28.0	91.0	231.	10.0	45.0	136.	325.
$\nabla\phi$	$P\times\phi$	0.60	0.34	0.27	0.19	0.64	0.59	0.51	0.43
$\nabla^2\phi$	$\frac{(P^2+P)}{2}\phi$	0.70	0.43	0.28	0.20	0.91	0.72	0.56	0.46

Table 5.4.30 gives effort ratios obtained from the results shown in figures 5.4.28 and 5.4.29. The effort ratios are defined by:

$E_{A,B}$ = the computational effort to evaluate A / the computational effort to evaluate B ...5.4.31.

5.5 Total computational effort.

The results obtained in the previous three sections of this chapter are summarised in this section.

One iteration in MAP requires, a function evaluation, a first derivative evaluation, and the solution of the LP problem. Thus an estimate of the total computational effort required by MAP to perform one iteration on a truss problem, is given by:

$$T_{5.5.1} = 423.6\,P^3 + 2383\,P^2 + 2678\,P + 635.9 \qquad ...5.5.1,$$

and on a plate problem is given by:

$$T_{5.5.2} = 372.8\,P^3 + 3755\,P^2 + 4018\,P - 26100 \qquad \text{...5.5.2.}$$

One search in MFD requires, a function evaluation, a first derivative evaluation, the solution of a LP problem and two more function evaluations on average, to locate the next set of constraints. Thus an estimate of the total computational effort required by MFD to perform one search on a truss problem, is given by:

$$T_{5.5.3} = 182.3\,P^3 + 2840\,P^2 + 7549\,P + 1908 \qquad \text{...5.5.3,}$$

and on a plate problem is given by:

$$T_{5.5.4} = 109.6\,P^3 + 2940\,P^2 + 13840\,P - 44370 \qquad \text{...5.5.4.}$$

One search in Newton's method with SUMT requires a function evaluation, a first and second derivative evaluation and 5.5 more function evaluations on average, to find the minimum along the direction. Thus an estimate of the total computational effort required by Newton's method to perform one search on a truss problem, is given by:

$$T_{5.5.5} = 29.0\,P^4 + 552.0\,P^3 + 3351\,P^2 + 11720\,P + 6812 \qquad \text{...5.5.5}$$

and on a plate problem, is given by:

$$T_{5.5.6} = 9.05\,P^4 + 1158\,P^3 + 6583\,P^2 + 13490\,P - 73670 \qquad \text{...5.5.6.}$$

If the objective function were quadratic, then Newton's method would require only one iteration to find its minimum.

One search in Fletcher-Powell's method with SUMT requires a function evaluation, a first derivative evaluation and 5.5 function evaluations, on average, to find the minimum along the direction. Thus, an estimate of the total computational effort required by the method to perform one search on a truss problem, is given by:

$$T_{5.5.7} = 135.9\,P^3 + 2047\,P^2 + 10600\,P + 6825 \qquad \ldots 5.5.7,$$

and on a plate problem, is given by:

$$T_{5.5.8} = 24.8\,P^3 + 2872\,P^2 + 25520\,P - 73660 \qquad \ldots 5.5.8.$$

Similarly, when Stewart's method with finite difference derivatives is used, the total computational effort to perform one search on a truss problem is given by:

$$T_{5.5.9} = 12.0\,P^4 + 156.9\,P^3 + 2712\,P^2 + 10400\,P + 6853 \qquad \ldots 5.5.9,$$

and on a plate problem, is given by:

$$T_{5.5.10} = 1.03\,P^4 + 86.2\,P^3 + 4535\,P^2 + 16690\,P - 56670 \qquad \ldots 5.5.10.$$

If the objective function were quadratic, then both Fletcher-Powell's and Stewart's methods would require no more than P iterations to find its minimum.

When Powell's method is used with SUMT, then the total computational effort to form and search along a direction on a truss problem, is given by:

$$T_{5.5.11} = 78.0\,P^3 + 1163\,P^2 + 9393\,P + 6869 \qquad \ldots 5.5.11,$$

and on a plate problem, is given by:

$$T_{5.5.12} = 6.70\,P^3 + 517\,P^2 + 25440\,P - 56650 \qquad \ldots 5.5.12.$$

Powell's method requires P or P+1 searches per iteration, and requires no more than P iterations to minimize a quadratic function.

For the truss problems and the plate problems respectively, figures 5.5.13 and 5.5.14 plot estimates of the computational effort required by each method to complete one iteration using an IBM 360/67 computer (as given by equations 5.5.1 to 5.5.12) against the number of design variables.

An iteration is defined as:

the solution of one LP problem when using MAP,

the solution of one-dimensional search when using MFD, N1, FP or ST, and

the solution of P one-dimensional searches when using PO.

Measurements of the actual computational effort used by each algorithm during one iteration are given in chapter 7.

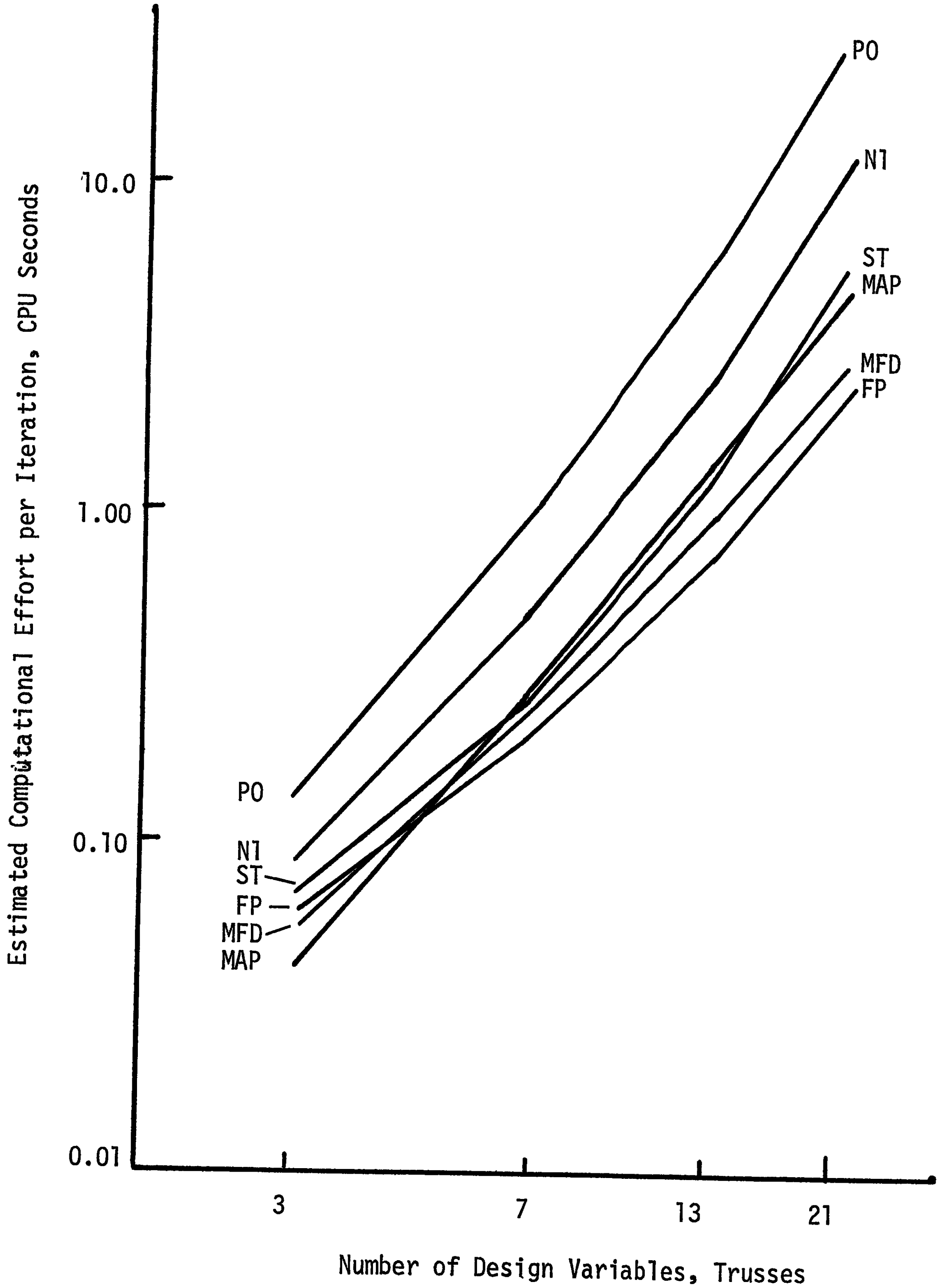
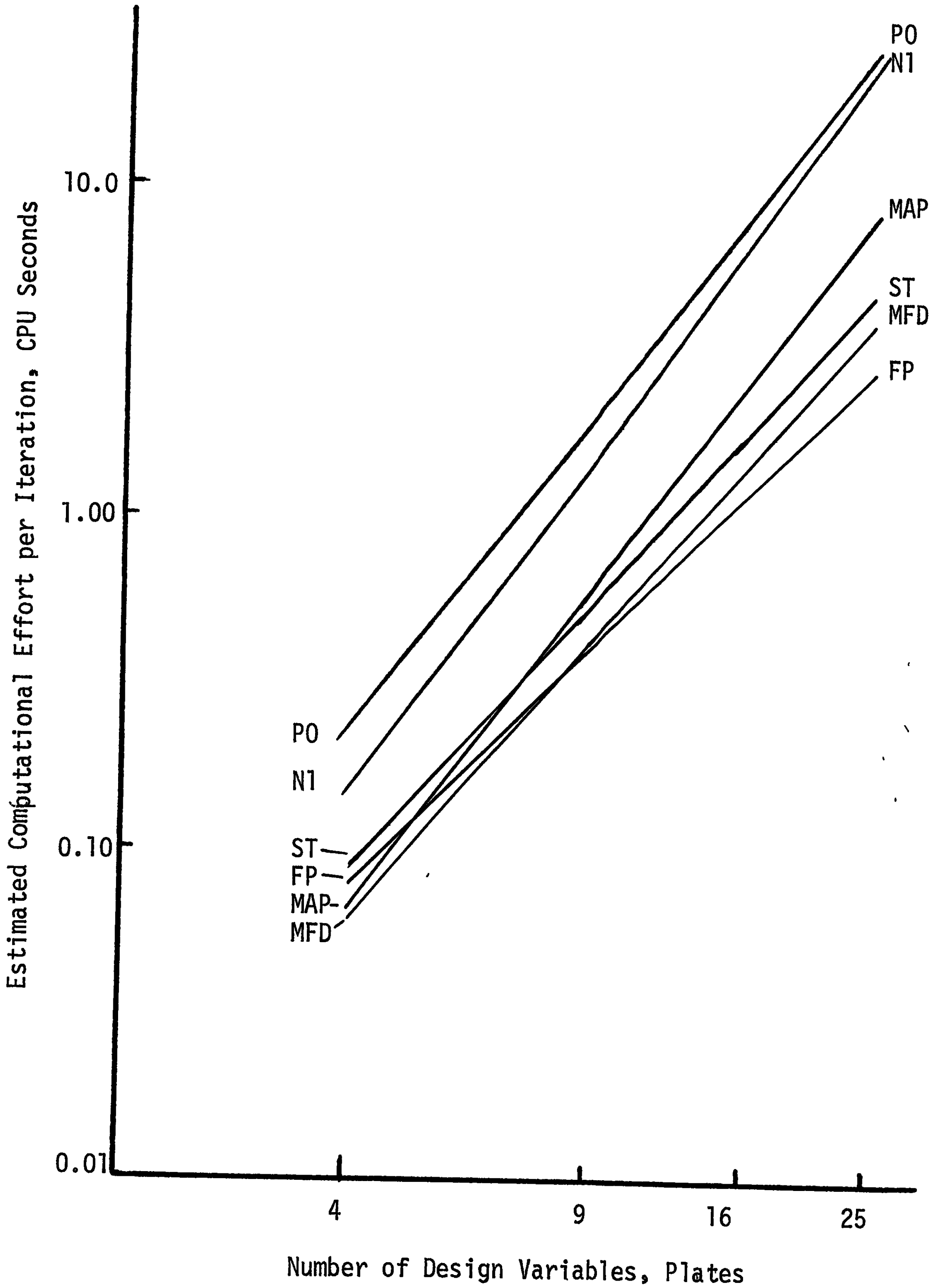

Estimated Computational Effort per Iteration, CPU Seconds
10.0
1.00
0.10
0.01
PO
N1
ST
MAP
MFD
FP
PO
N1
ST
FP
MFD
MAP
3
7
13
21
Number of Design Variables, Trusses

FIGURE 5.5.13



PO
N1
MAP
ST
MFD
FP
10.0
1.00
0.10
0.01
PO
N1
ST
FP
MAP
MFD
Estimated Computational Effort per Iteration, CPU Seconds
4
9
16
25
Number of Design Variables, Plates

FIGURE 5.5.14

CHAPTER 6

TEST PROBLEM DATA

6.1 Description of the tests.

Chapter 5 developed estimates of the computational effort required by each of the algorithms on the two types of structural problem under consideration when using an IBM 360/67 computer. This chapter gives details of test problems investigated using an IBM 370/145 computer to ascertain the actual computational effort used by each of the algorithms. The following results were recorded:

1. the number of
   a. one-dimensional searches,
   b. function evaluations, and
   c. derivative evaluations;
2. the CPU time expended in
   a. evaluating functions,
   b. evaluating derivatives, and
   c. performing those operations required by the optimisation algorithms; and
3. the value of
   a. the objective function and
   b. the structural weight.

The CPU times, measured using a system subroutine, do not include CPU effort expended performing input/output operations. The test structures used are:

3, 7, 13 and 21 member pin-jointed plane trusses, and

4, 9, 16 and 25 node idealization plane stress plates,

all subject to two load cases, with upper and lower bounds on stress and design variable values. Data for the structures are given in the following sections of this chapter.

The optimisation algorithms used are summarized in section 7 of chapter 2 and are detailed in chapter 3. Selection is made of arbitrary coefficients and other parameters required by the algorithms in section 3 of this chapter.

6.2 Test structure data

The trusses used in this study are similar to one investigated by Schmit[29]. The design variables are the member cross-sectional areas. The configurations of the test trusses are shown in figure 6.2.1 and have the following common data:

initial cross-sectional area of all members = 1.0,

load case 1, P_1 = 15.0, P_2 = 25.9808,

load case 2, P_1 =-20.0, P_2 = 0.0,

Young's modulus of elasticity = 1.0,

density = 1.0,

$t_{max\ j}$ = 20.0, ...6.2.2

$t_{min\ j}$ = 0.01,

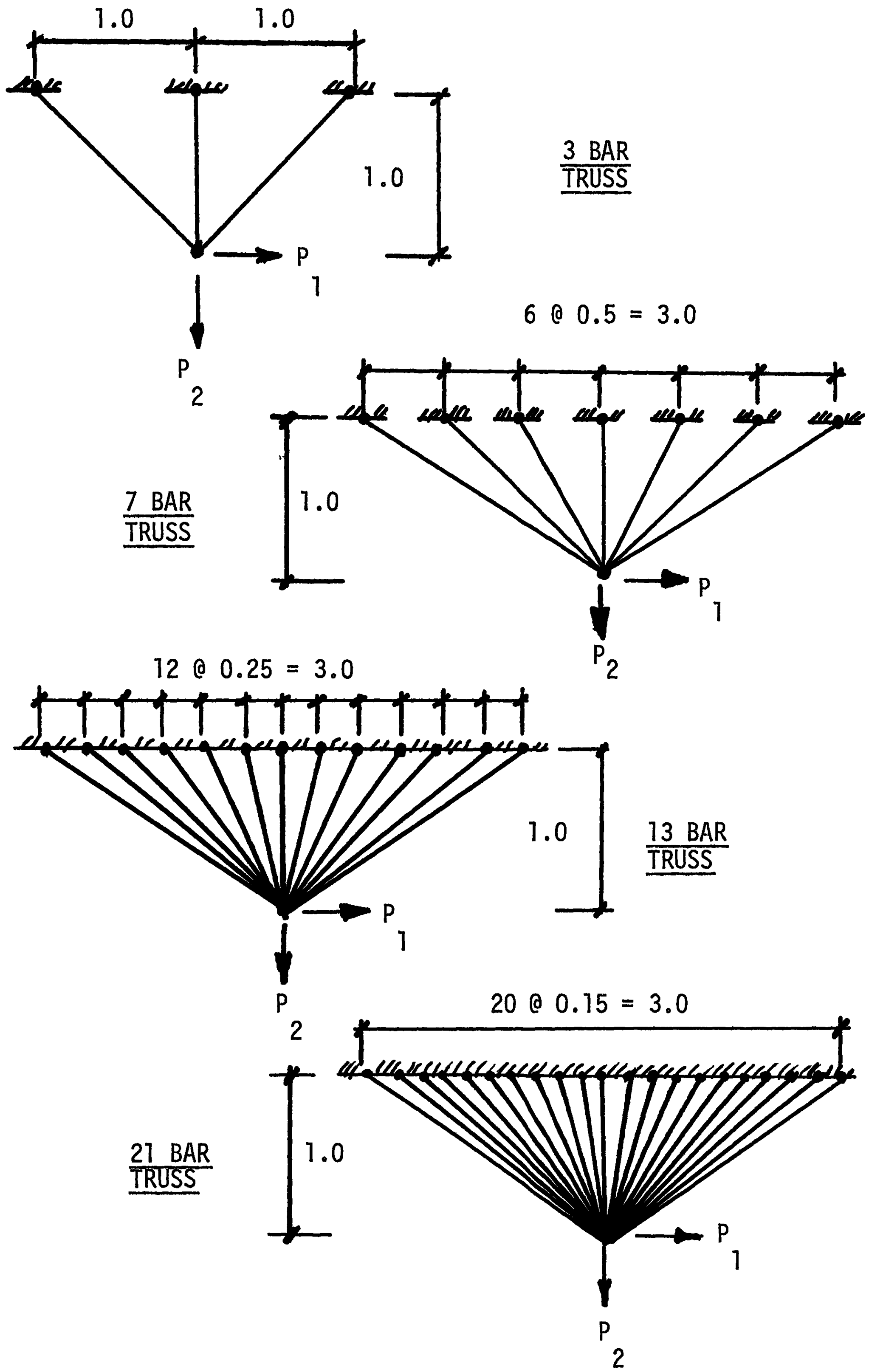
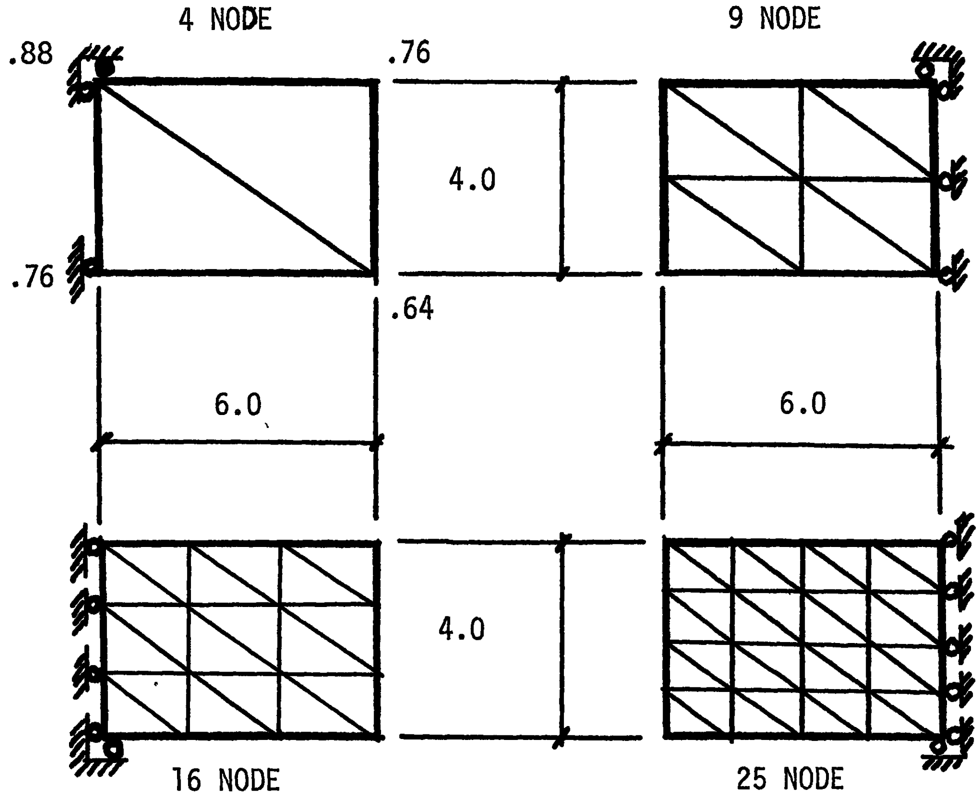

1.0
1.0
1.0
3 BAR TRUSS
P1
P2
6 @ 0.5 = 3.0
7 BAR TRUSS
1.0
P1
P2
12 @ 0.25 = 3.0
1.0
13 BAR TRUSS
P1
P2
20 @ 0.15 = 3.0
21 BAR TRUSS
1.0
P1
P2

FIGURE 6.2.1


4 NODE
9 NODE
.88
.76
.76
.64
4.0
6.0
6.0
4.0
16 NODE
25 NODE

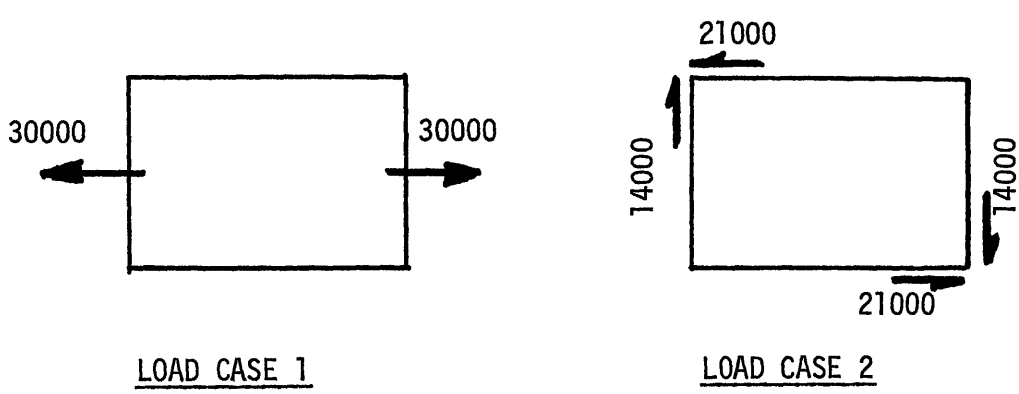

30000
30000
21000
14000
14000
21000
LOAD CASE 1
LOAD CASE 2

FIGURE 6.2.3

$\sigma_{max\ qs} = 20.0,$

$\sigma_{min\ qs} = -15.0.$

The plane stress plates used in this study are rectangular and are subject to two load cases, one of pure tension and one of pure shear. The nodal thicknesses of the finite element idealization are the design variables. The plates have the configurations shown in figure 6.2.3 and have the following common data:

the initial thicknesses of the nodes are determined from linear interpolation using the initial thicknesses of the nodes of the 4 node plate,

Young's modulus of elasticity = 10,000,000.0,

Poisson's ratio = 0.3,

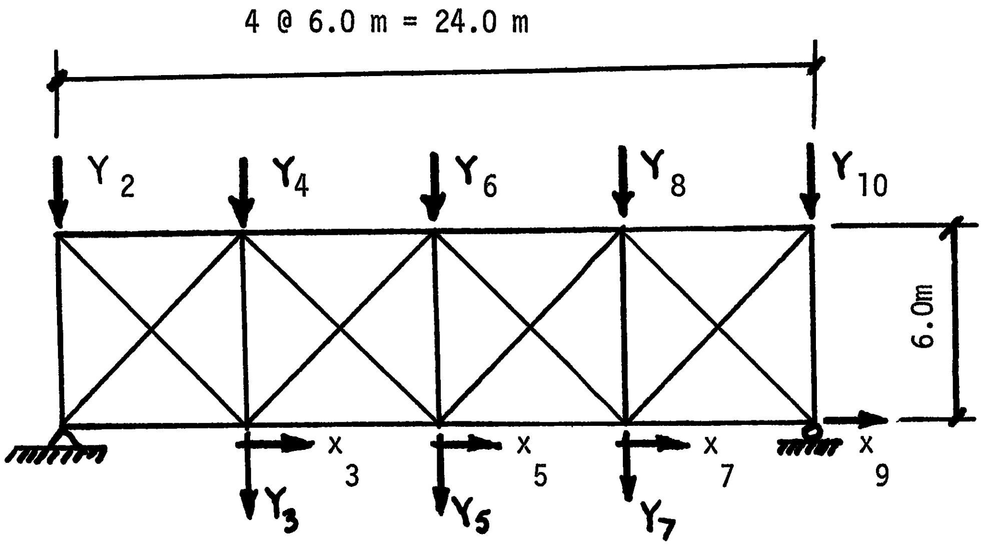
density = 2.0,

$t_{max\ j} = 1.0,$ ...6.2.4

$t_{min\ j} = 0.25,$

$\sigma_{max\ qs} = 15,000.0 = -\sigma_{min\ qs}.$

Figure 6.2.5 shows the configuration of a 21 bar bridge which was also used to test the optimisation algorithms. The bridge was subjected to one dead load and four live load cases. The live loadings are of the type imposed by vehicles on a bridge truss. Table 6.2.6 gives load data. Other pertinent data are:



21 BAR BRIDGE

FIGURE 6.2.5

initial area of all members = 95.0 cm^2,

initial weight = 106.7 kN = 10.86Mg,

Young's modulus of elasticity = 21,000 kN/cm^2,

density = 7.698×10^{-5} kN/cm^3 = 0.785×10^{-5} Mg/cm^3,

$t_{max\ j}$ = 100 cm^2,

$t_{min\ j}$ = 10 cm^2,

$\sigma_{max\ qs}$ = 16.5 kN/cm^2, and $\sigma_{min\ qs}$ = - 12.0 kN/cm^2.

Results for this structure are given in the appendices.

Table 6.2.6: Loadings on the 21 bar bridge (kN)

Load	Dead load	Live Load load case 1	load case 2	load case 3	load case 4
X_3	-	- 40	- 40	-	-
X_5	-	-	- 40	+ 40	-
X_7	-	-	-	+ 40	+ 40
X_9	-	-	-	-	+ 40
Y_2	10	-	-	-	-
Y_3	55	+ 200	+ 200	-	-
Y_4	15	-	-	-	-
Y_5	55	-	+ 200	+ 200	-
Y_6	15	-	-	-	-
Y_7	55	-	-	+ 200	+ 200
Y_8	15	-	-	-	-
Y_{10}	10	-	-	-	-

6.3 Optimisation algorithm data

Operational characteristics of optimisation algorithms are dictated by control parameters and/or arbitrary coefficients. Values for the arbitrary coefficients (α in equation 3.2.2 for MAP, c_i in problem formulation 3.3.19 for the MFD, and c in equation 3.4.2 for the SUMT) are selected in section 4 of this chapter. Values for the control parameters required by the optimisation program used in this study are given below.

The control parameters are used by the optimisation program to determine when control should be returned from a subroutine to the calling subroutine or program and to determine when the optimisation should be terminated. The control parameters are set in the main program or are read as data input. The algorithm for the main program used is:

i. read in structural data and optimisation data; ...6.3.1

ii. set values for the control parameters for the algorithms on this iteration of the main program;

iii. go to the optimisation algorithm and on return from the algorithm go to iv;

iv. record results (section 1 of this chapter);

v. if the optimisation should be terminated, report results and terminate; if the optimisation should not be terminated go to ii.

In step i, the following optimisation data are input:

1. the values of the arbitrary coefficients;
2. the relative accuracy of
   a. number representation and
   b. function and derivative evaluations;
3. the minimum allowable relative rates of
   a. reduction in weight,
   b. reduction in objective function, and
   c. change in all the design variable values;
4. the resolution of design points; and
5. the maximum number of main program iterations allowed.

The data items 2a and 2b are used by many of the algorithms to generate the test values in the algorithms. The relative accuracy of number representation depends on the absolute magnitude of the number represented, but was taken to be an average value of 0.000 000 1 for the computers used in the tests. Preliminary tests showed that the relative accuracy of function and derivative evaluations was approximately 0.000 001.

Data item 3a is required in step v of the main program algorithm. The program is terminated if the actual relative reduction in weight during the latest main program iteration is less than the value 3a. A value of 0.000 010 per main program iteration was used.

Data items 3b and 3c are used in step iii by the optimisation algorithms to transfer control to the main program. Thus, if the relative reduction in the objective function and relative change in all the design variables are less than the values 3b and 3c, then the optimisation algorithms return control to the main program. A value of 0.001 per optimisation algorithm iteration was used for both 3b and 3c.

Data item 4 is used by the one-dimensional search algorithms. A value of 0.001 was used for the resolution of design points. The maximum number of main program iterations, data item 5, was set at 7.

The control parameters set in step ii are: the maximum number of quadratic fits allowed in each one-dimensional search and the maximum number of optimisation algorithm iterations allowed per main program iteration.

Table 6.3.2: Maximum number of algorithm iterations allowed

Algorithm	Maximum number of iterations per main program iteration allowed.
MAP	1 + i/3 *
MFD	4 + i/2
PO	
ST	(2 + i/2)P
FP	
N1	2 + i/2
N2	(1 + P(3 + i))/2

* where i is the iteration number of the main program.

In step ii, the maximum number of quadratic fits allowed per one-dimensional search was set at $(6 + i/3)$ for all the algorithms except MAP.

In step iii of the main program algorithm, the optimisation algorithms return to the main program when the maximum number of iterations given in table 6.3.2 is exceeded.

The results in step iv of the main program algorithm are listed in section 1 of this chapter. When MAP is being used, the weight of a feasible design obtained from an infeasible solution is also recorded. The feasible design is determined by multiplying all the design variables of the infeasible solution, by the ratio of the stress which violates the allowable stresses by the greatest amount to the appropriate allowable stress. The weight obtained from the feasible design is called the 'scaled weight' in the following chapters.

In step v of the main program algorithm, the following tests are made for termination:

i. terminate for all algorithms except MAP, if the weight has increased;

ii. terminate if the design is feasible or acceptably infeasible, and if the relative rate of change in weight during the latest main program iteration is less than the test value data item 3a;

iii. terminate if the number of main program iterations exceeds the test value, data item 5.

In test ii, a design is considered acceptably infeasible providing none of the stress constraints are violated by more than 0.000 001 $\sigma_{max\ qs}$.

6.4 Optimisation algorithm arbitrary coefficients.

Preliminary computer runs were made to establish suitable values for the arbitrary coefficients of the optimisation algorithms. The three bar and seven bar trusses were used as the test structures for these runs.

For the three and seven bar trusses respectively, figures 6.4.1 and 6.4.2 plot weight against CPU time for the values of the MAP arbitrary coefficient α from 0.10 to 0.40. From the results shown, α was selected as 0.20 for all the computer runs reported in chapter 7.

For the three and seven bar trusses respectively, figures 6.4.3 and 6.4.4 plot weight against the number of derivative evaluations for the values of the MFD arbitrary coefficients c_i (for all i) from 0.0001 to 10.0. From the results shown in these figures, c_i for all i was set at 0.10 for all the computer runs reported in chapter 7.

For the three bar truss only, figures 6.4.5 to 6.4.24 plot weight against CPU time for the values of the SUMT arbitrary coefficient c from 1/10 to 1/320. For data items 3b and 3c of section 3 of this chapter, figures 6.4.5 to 6.4.8 have the values of 0.005, 0.001, 0.0002 and 0.00004 respectively on Newton(1)'s method. Similarly, figures 6.4.9 to 6.4.12, 6.4.13 to

6.4.16 and 6.4.17 to 6.4.20 have the above values on Fletcher-Powell's, Stewart's and Powell's methods respectively. For Newton(1)'s, Fletcher-Powell's, Stewart's and Powell's methods respectively, figures 6.4.21 to 6.4.24 have, for data items 3b and 3c, the initial value of 0.001, which is then reduced, as recommended by Moe[22], by a constant factor of 0.4 on each succeeding SUMT iteration. From the results shown in figures 6.4.5 to 6.4.24, an efficient and consistently effective choice for the value of the SUMT arbitrary coefficient c is 1/160. This value was used for the computer runs reported in chapter 7. Figures 6.4.5 to 6.4.24 also verify that the value of 0.001 for data items 3b and 3c of section 3 is efficient and that the scheme suggested by Moe does not seem to offer significant computational advantages.

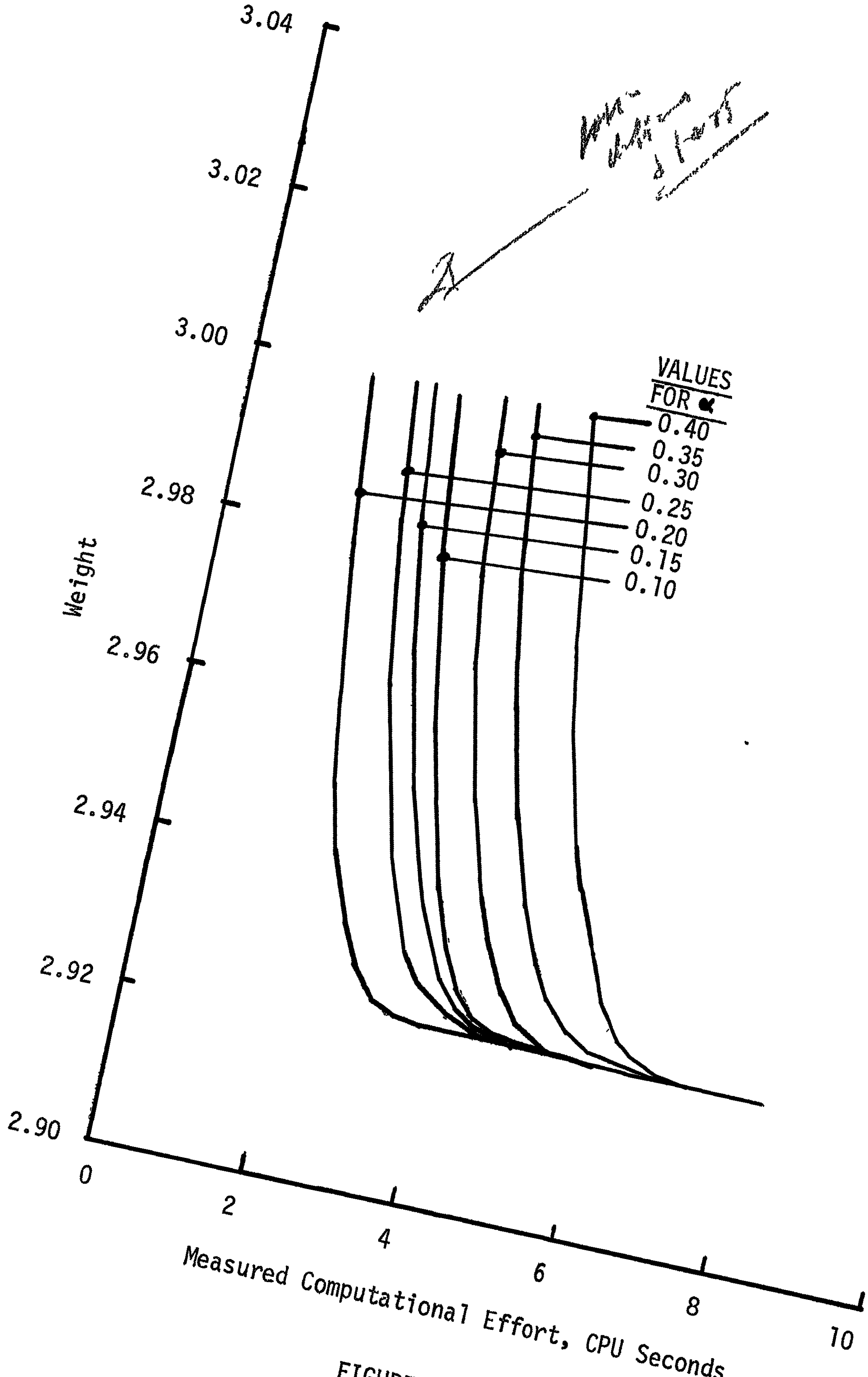

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
0
2
4
6
8
10
Measured Computational Effort, CPU Seconds
VALUES FOR α
0.40
0.35
0.30
0.25
0.20
0.15
0.10

FIGURE 6.4.1

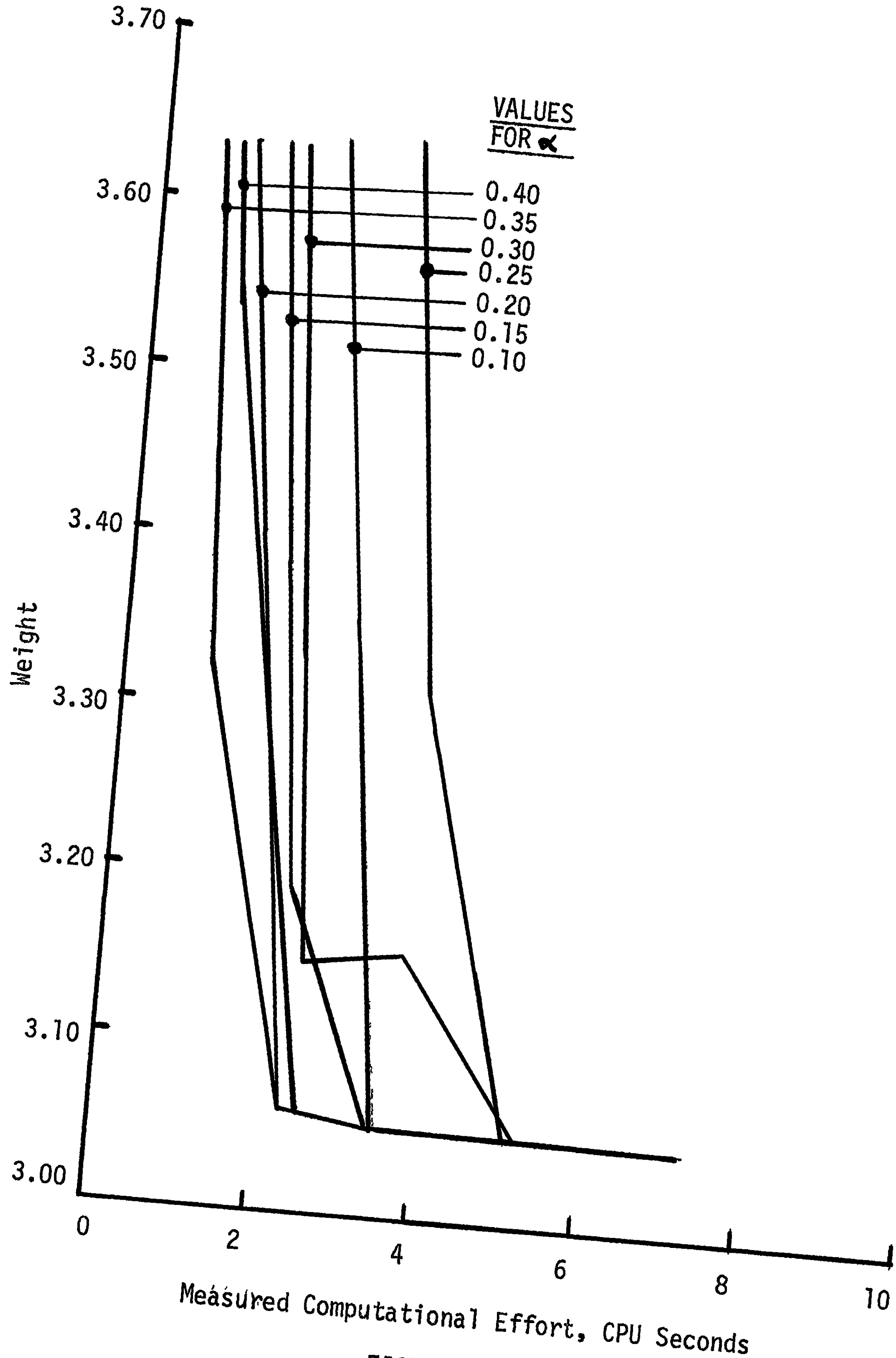

VALUES FOR α
0.40
0.35
0.30
0.25
0.20
0.15
0.10
3.70
3.60
3.50
3.40
3.30
3.20
3.10
3.00
Weight
0
2
4
6
8
10
Measured Computational Effort, CPU Seconds

FIGURE 6.4.2

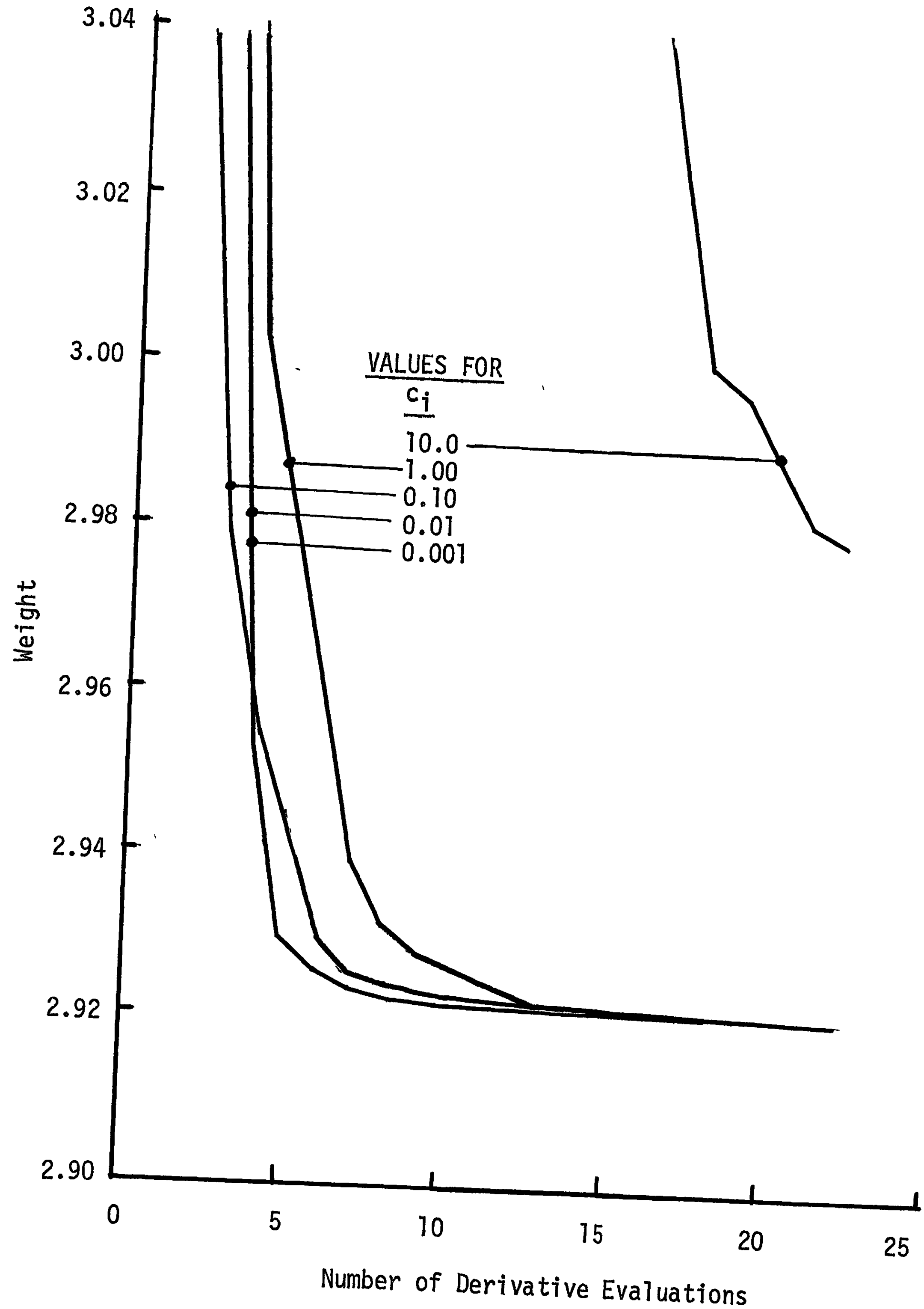

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR
c_i
10.0
1.00
0.10
0.01
0.001
0
5
10
15
20
25
Number of Derivative Evaluations

FIGURE 6.4.3

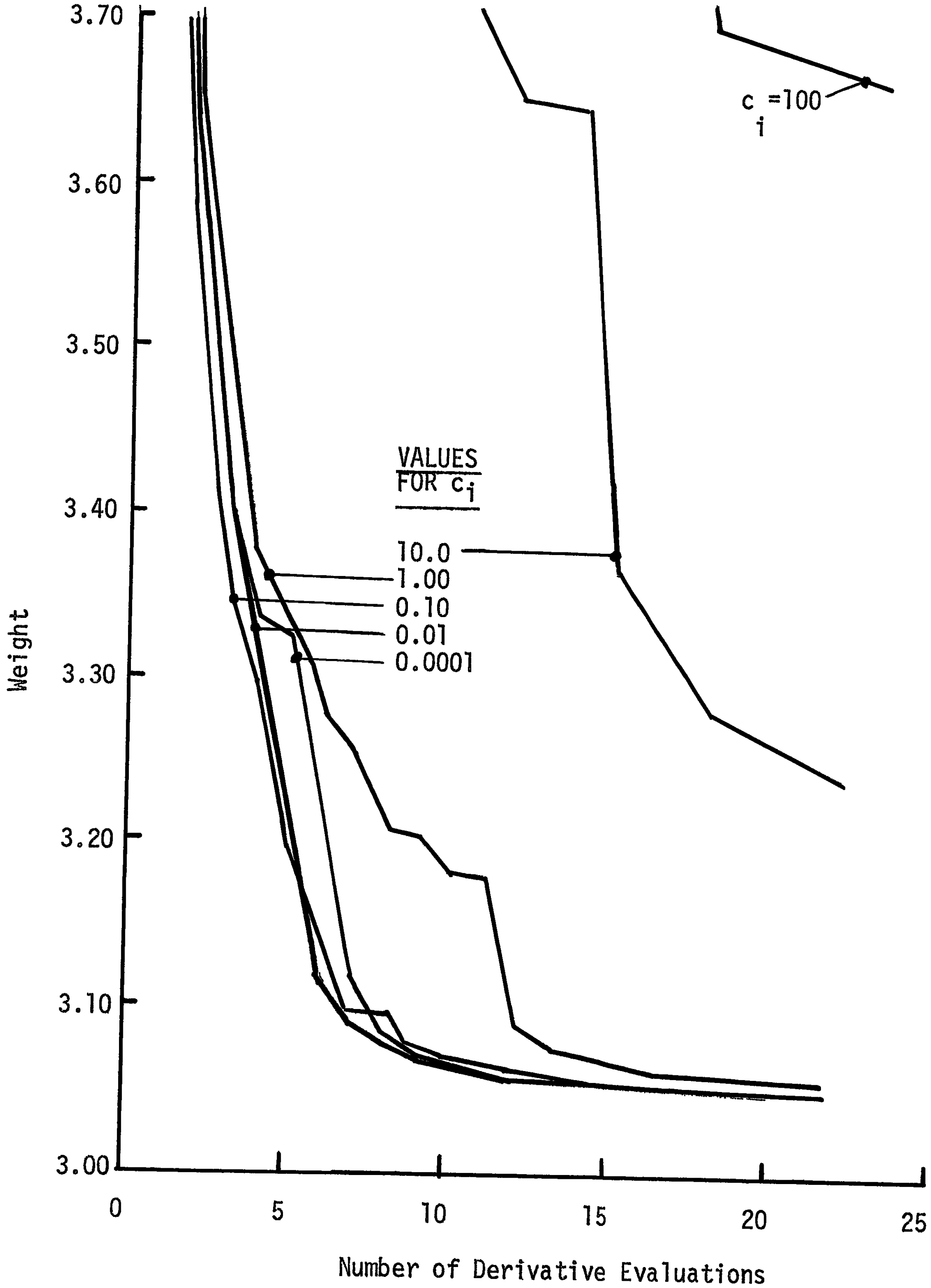

3.70
3.60
3.50
3.40
3.30
3.20
3.10
3.00
Weight
0
5
10
15
20
25
Number of Derivative Evaluations
VALUES FOR c_i
10.0
1.00
0.10
0.01
0.0001
c_i =100

FIGURE 6.4.4

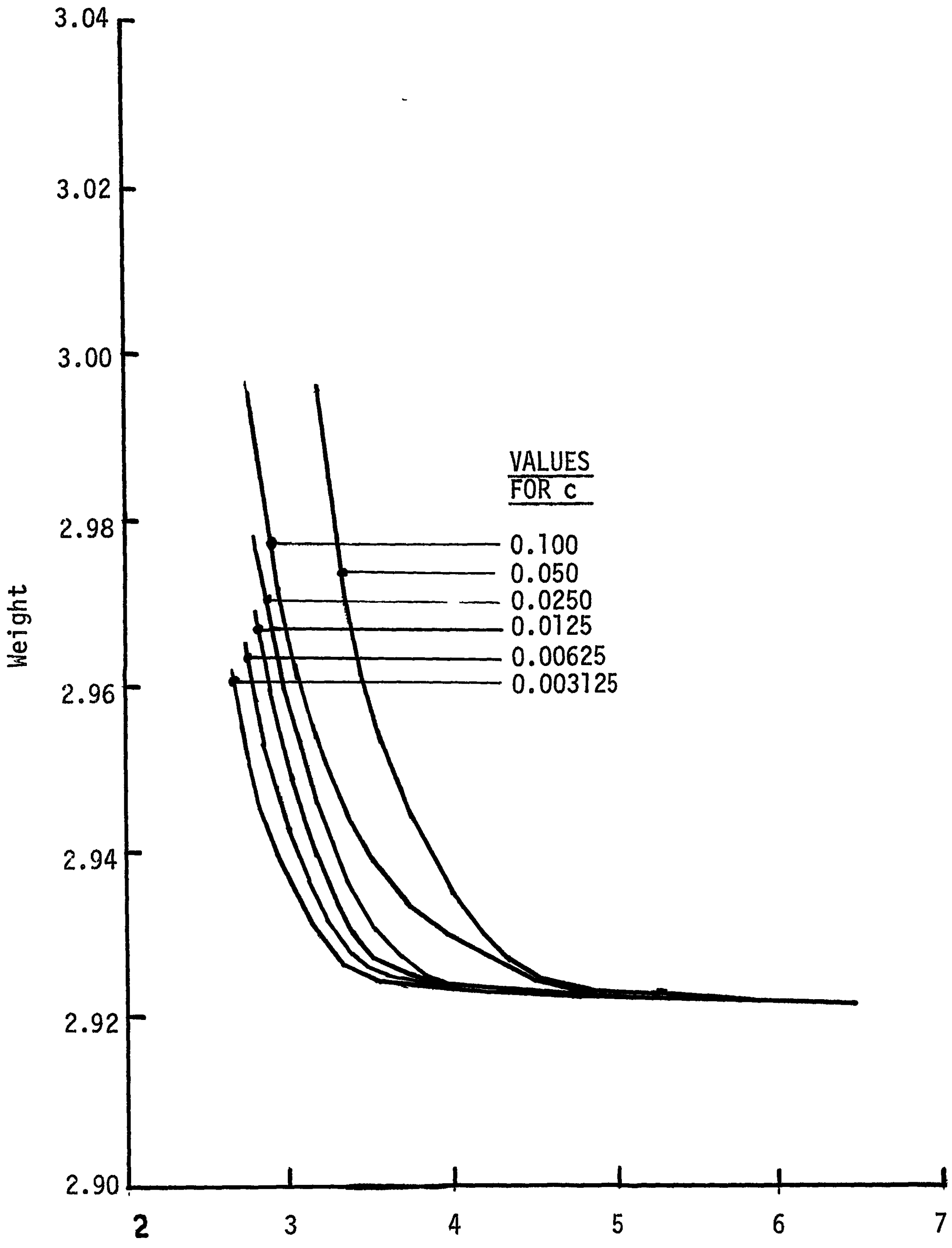

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.100
0.050
0.0250
0.0125
0.00625
0.003125
2
3
4
5
6
7
Measured Computational Effort, CPU Seconds

FIGURE 6.4.5

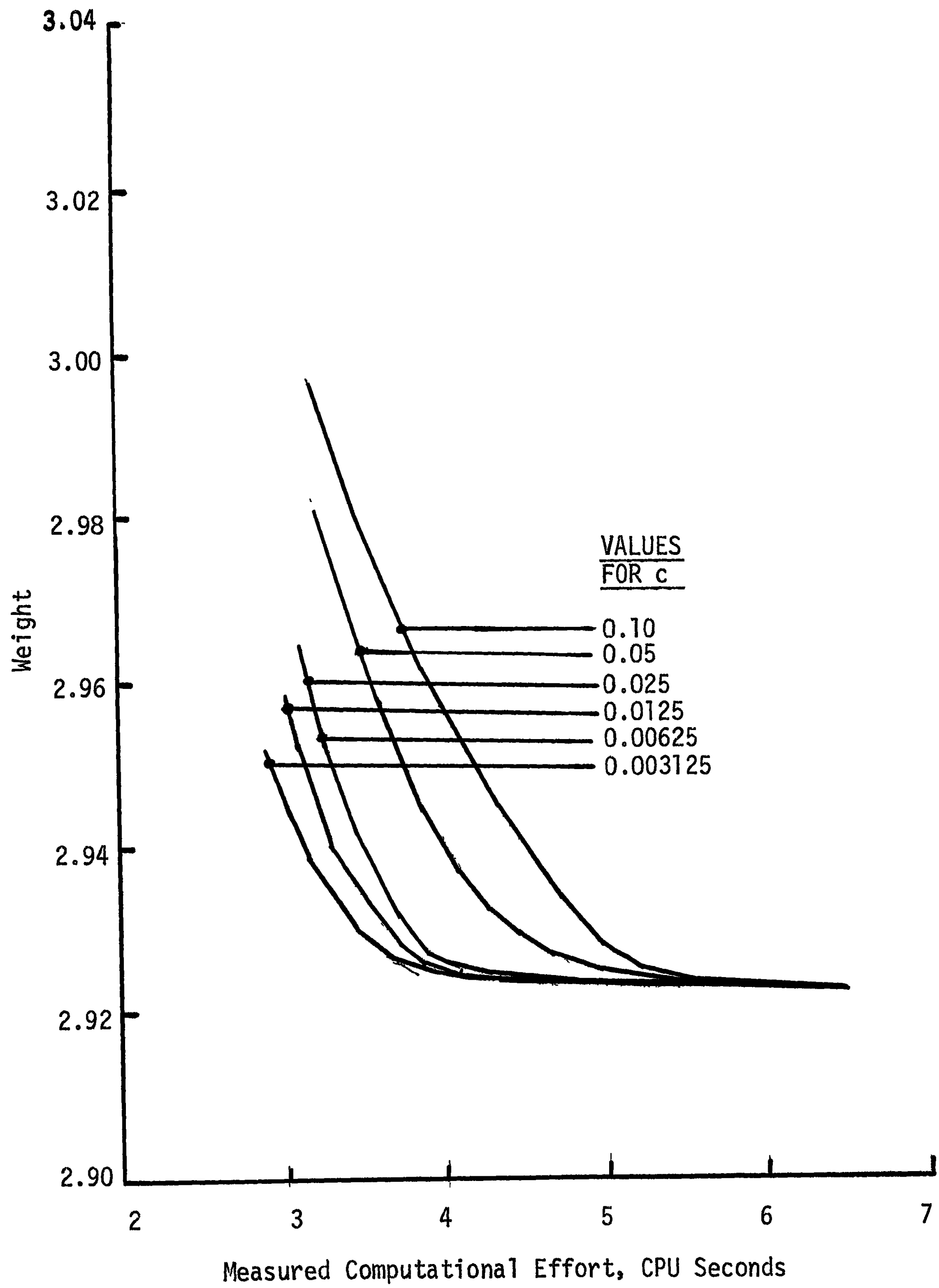

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125
2
3
4
5
6
7
Measured Computational Effort, CPU Seconds

FIGURE 6.4.6

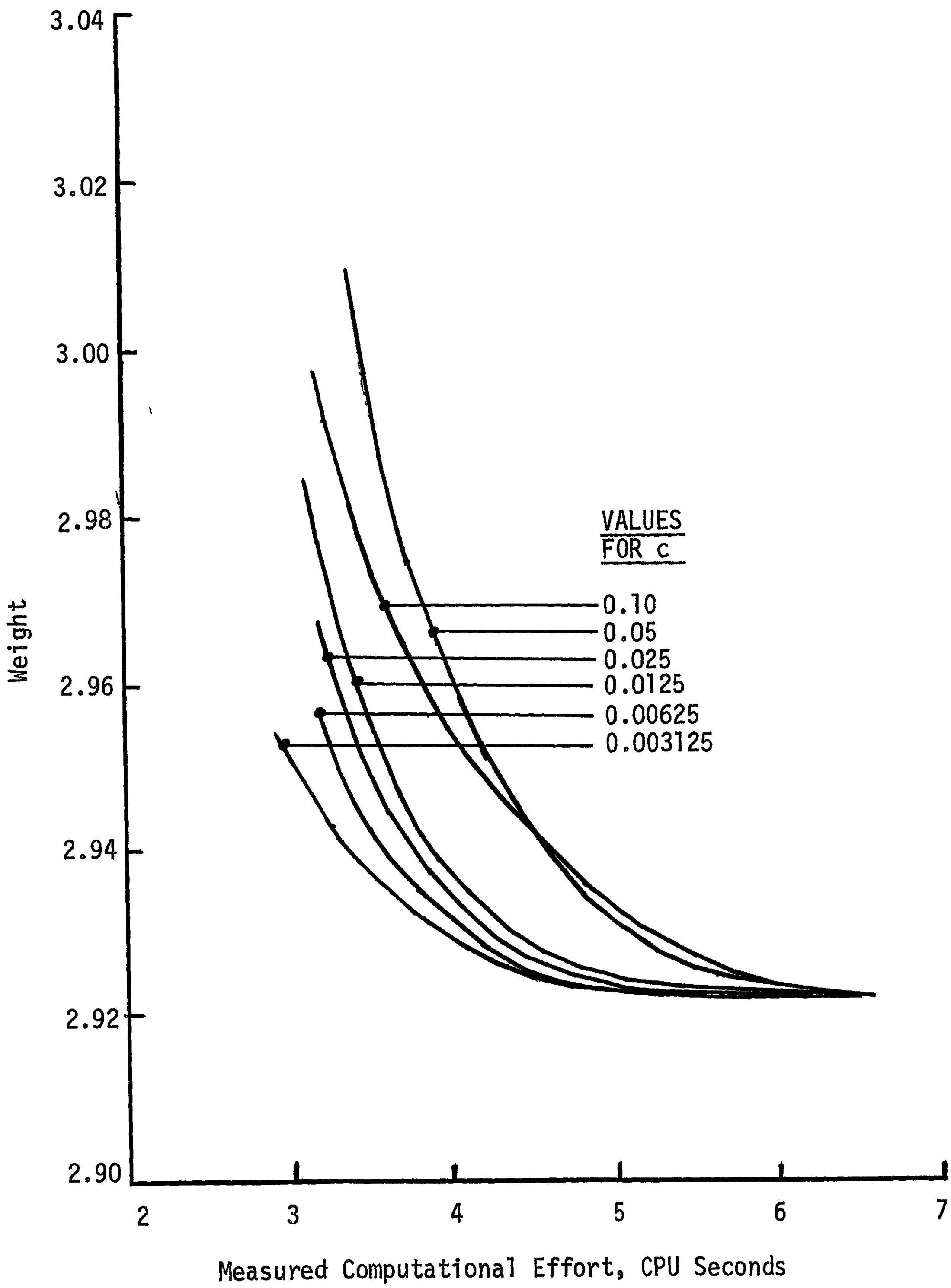

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125
2
3
4
5
6
7
Measured Computational Effort, CPU Seconds

FIGURE 6.4.7

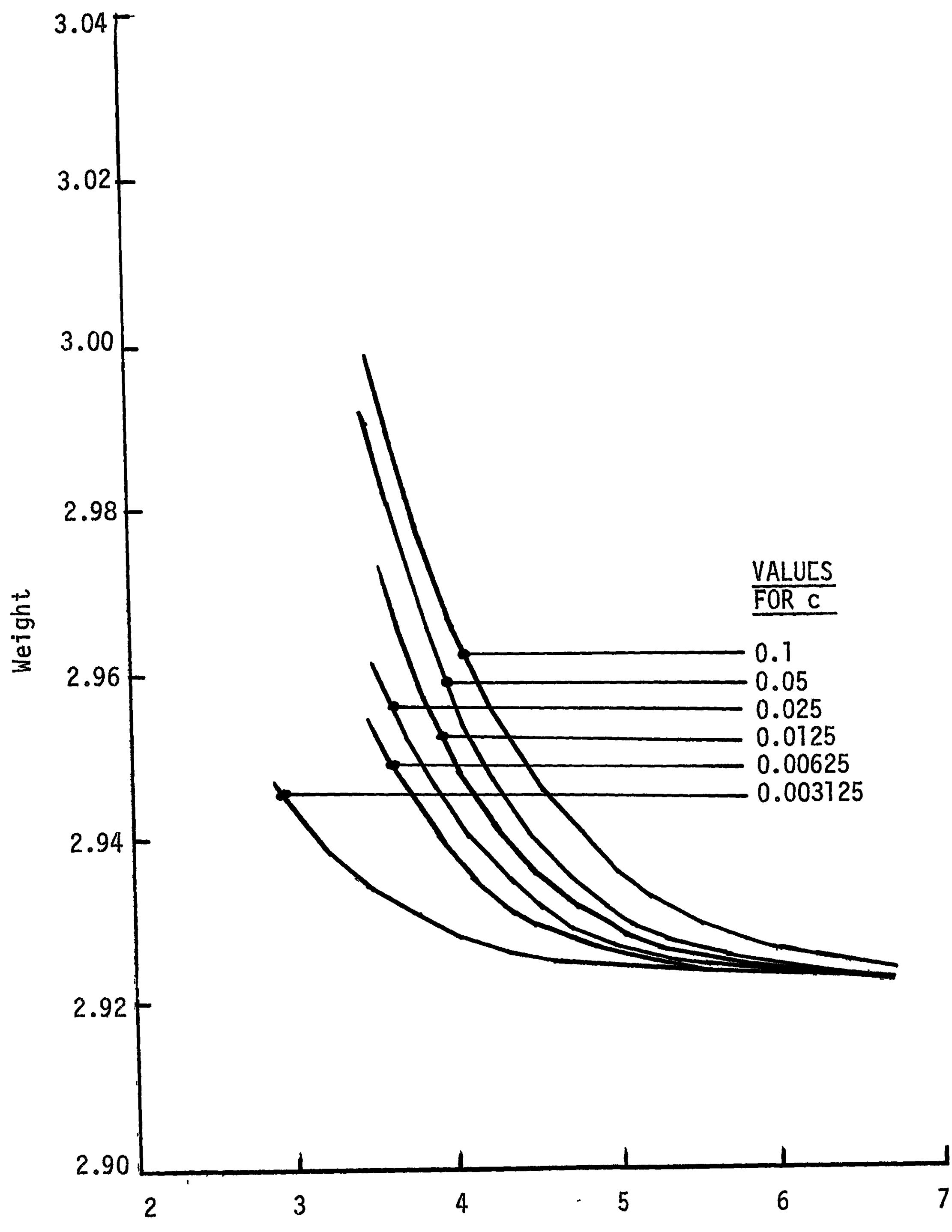

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
2
3
4
5
6
7
Measured Computational Effort, CPU seconds.
VALUES FOR c
0.1
0.05
0.025
0.0125
0.00625
0.003125

FIGURE 6.4.8

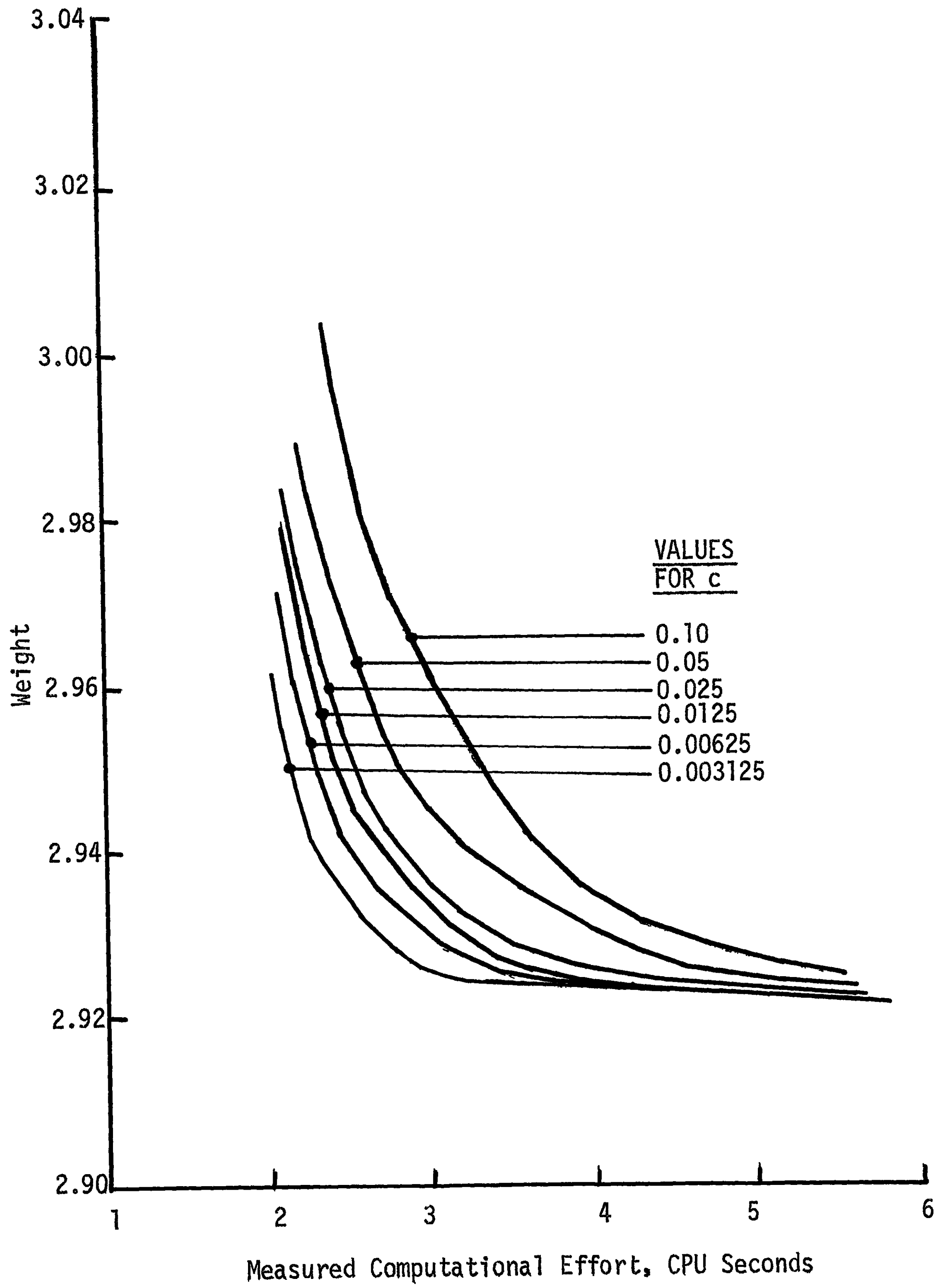
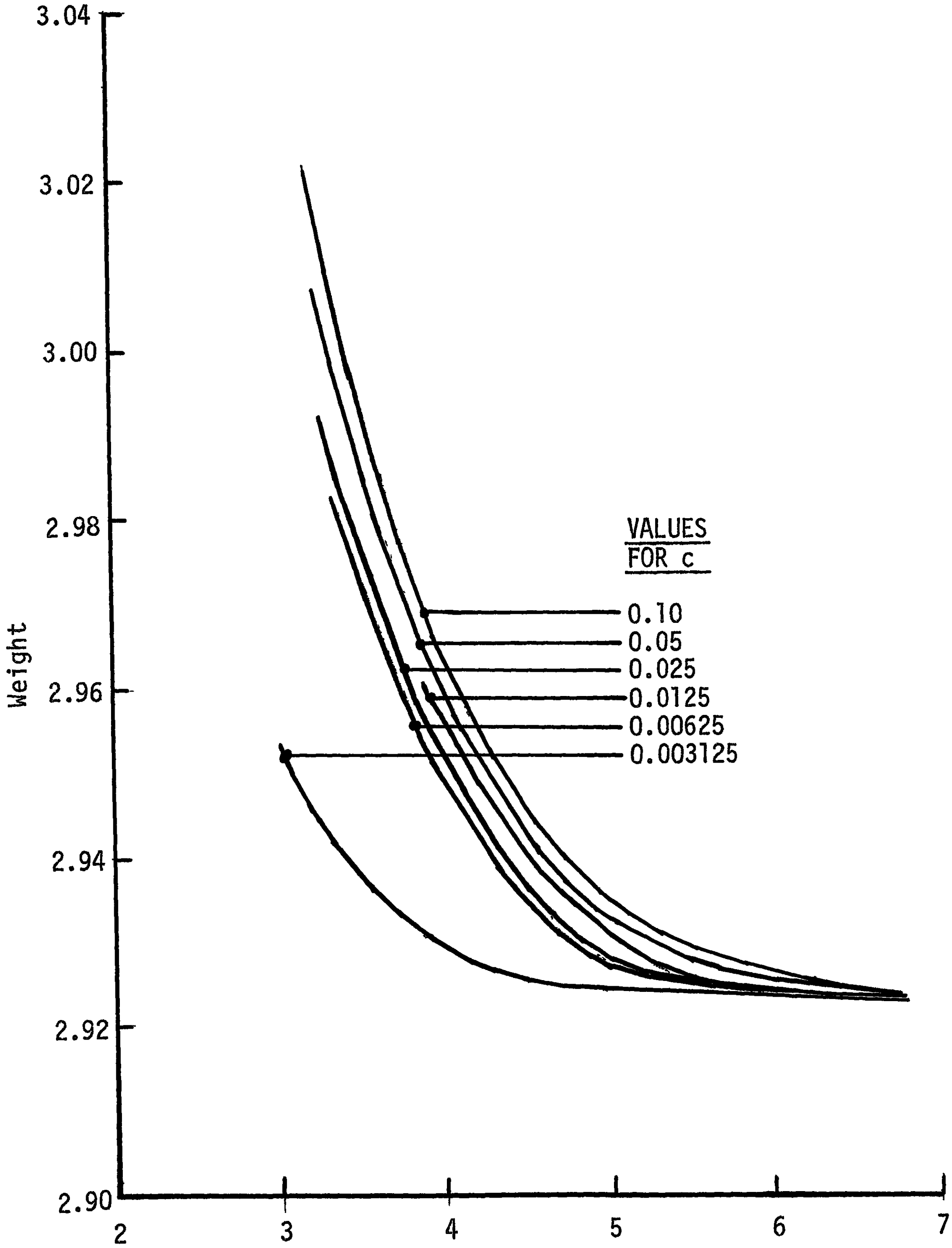

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125
1
2
3
4
5
6
Measured Computational Effort, CPU Seconds

FIGURE 6.4.9



Measured Computational Effort, CPU Seconds

FIGURE 6.4.10

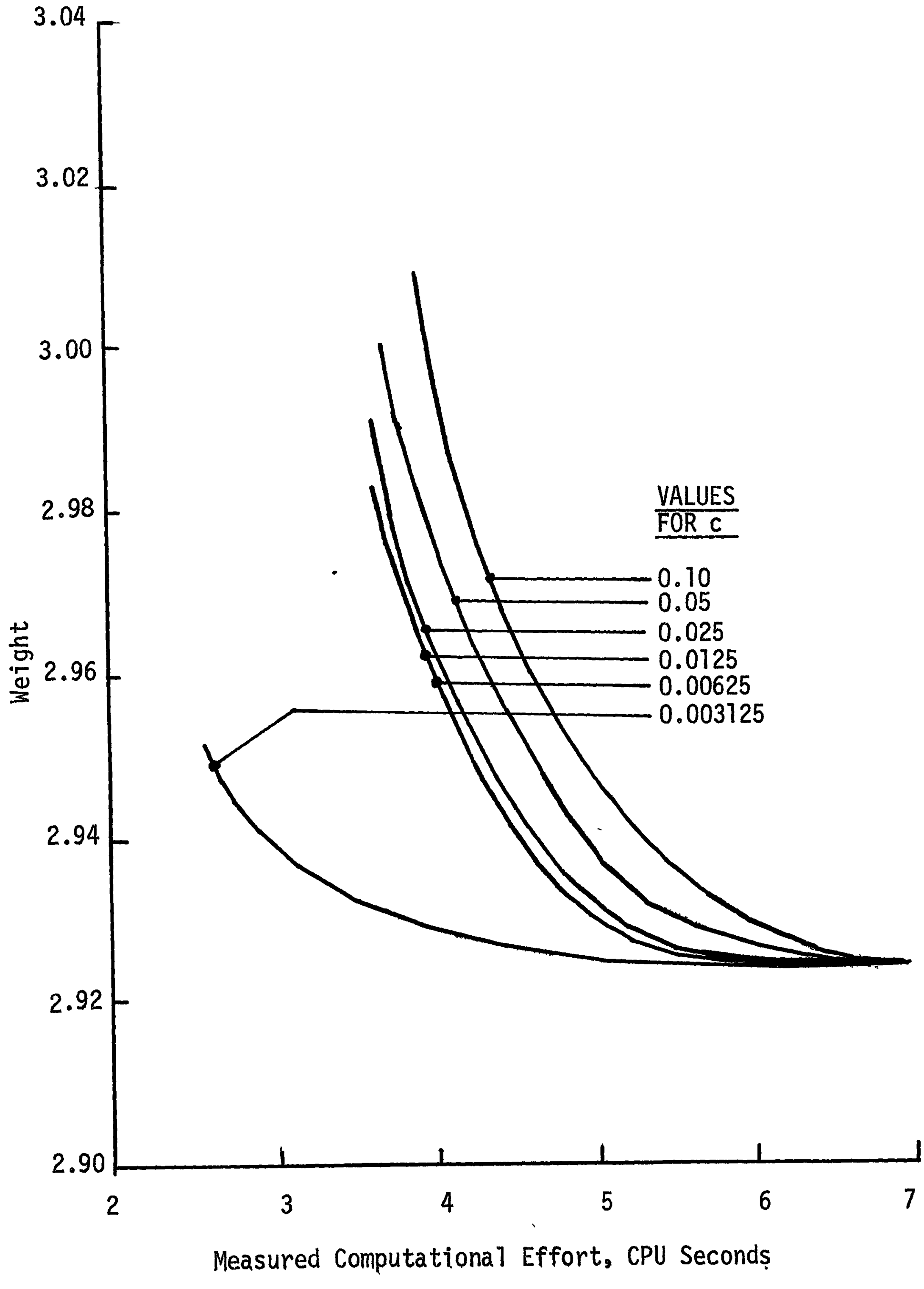

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125
2
3
4
5
6
7
Measured Computational Effort, CPU Seconds

FIGURE 6.4.11

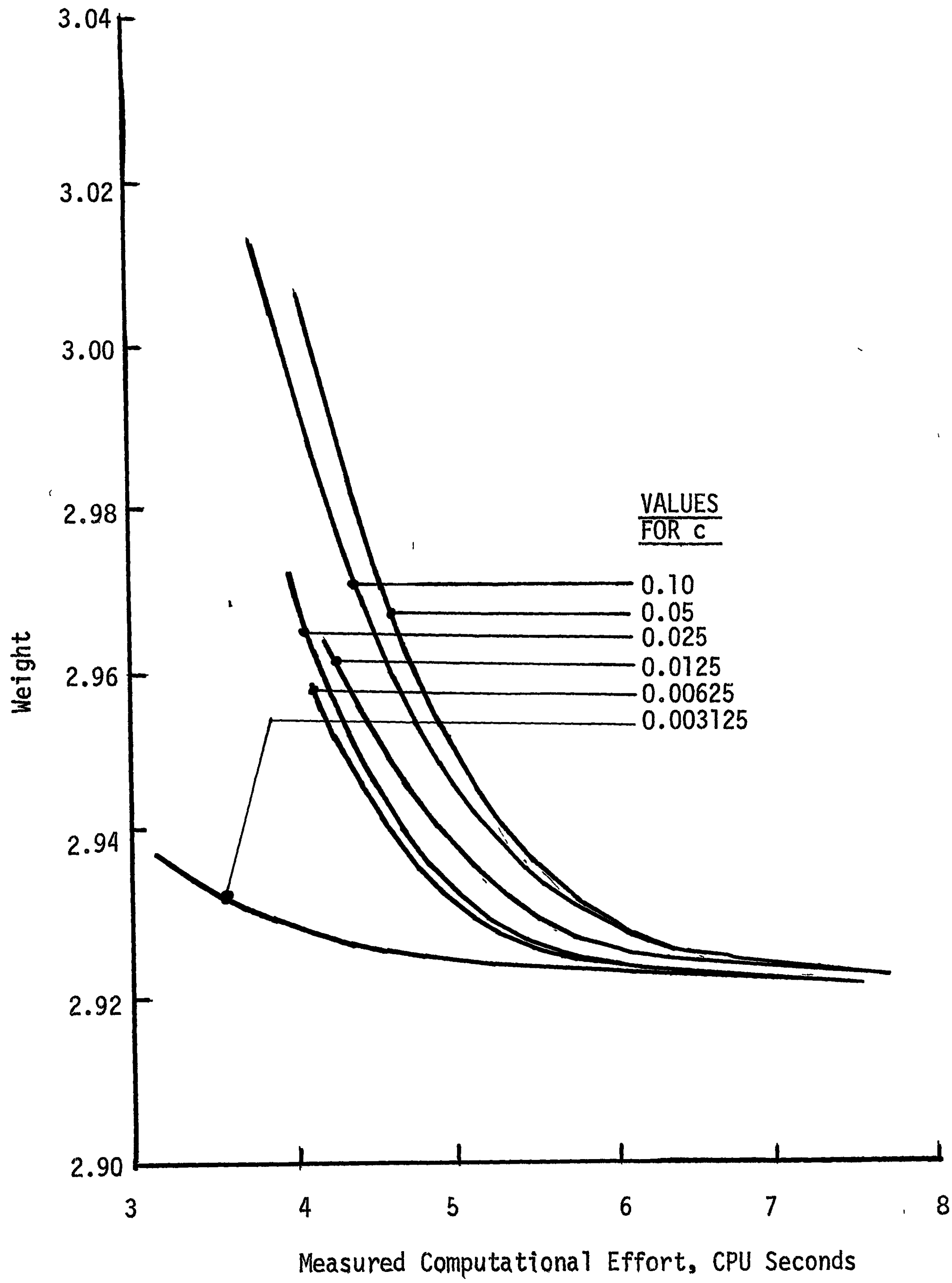
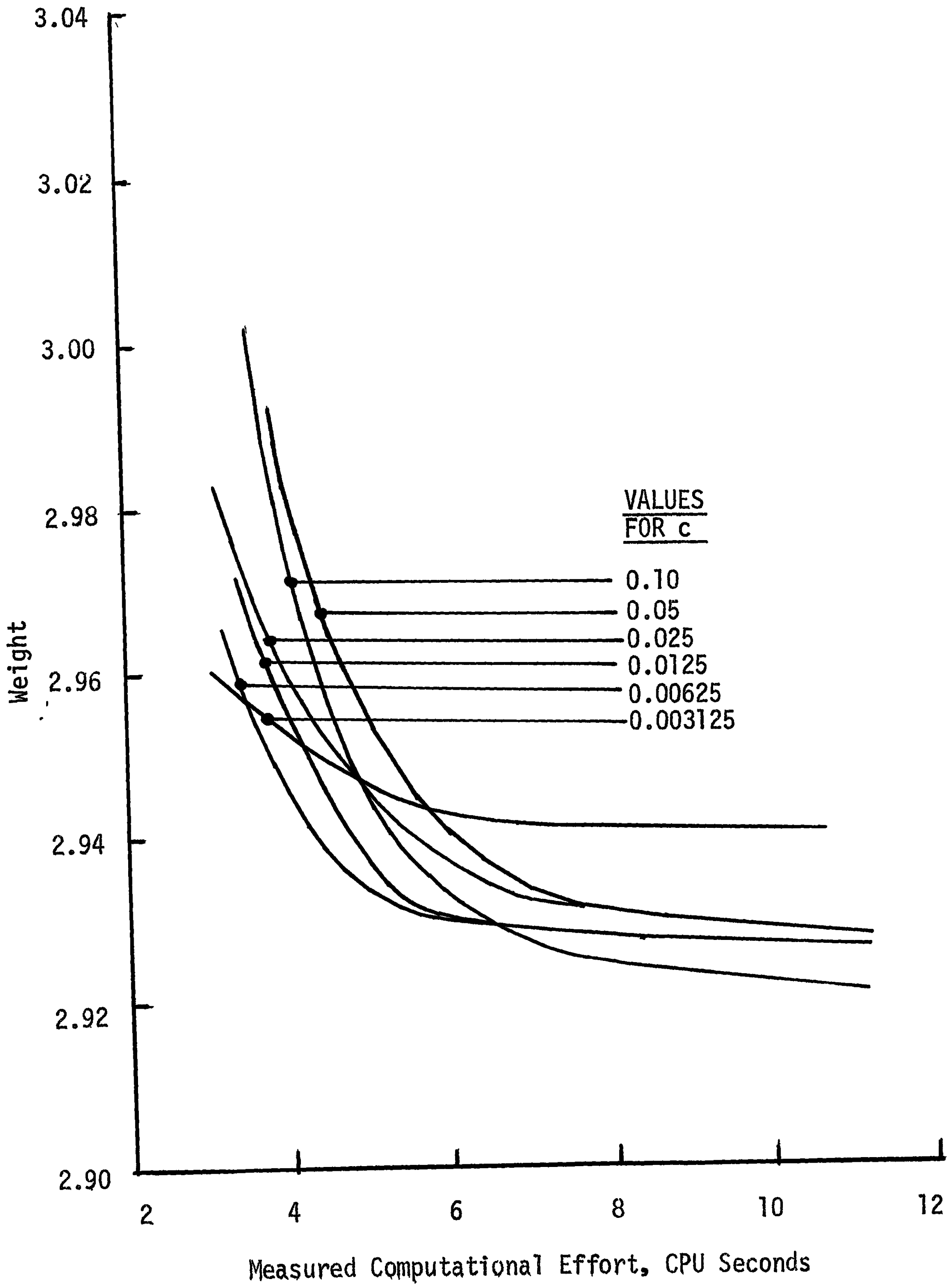

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125
3
4
5
6
7
8
Measured Computational Effort, CPU Seconds

FIGURE 6.4.12



3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
2
4
6
8
10
12
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125
Measured Computational Effort, CPU Seconds

FIGURE 6.4.13

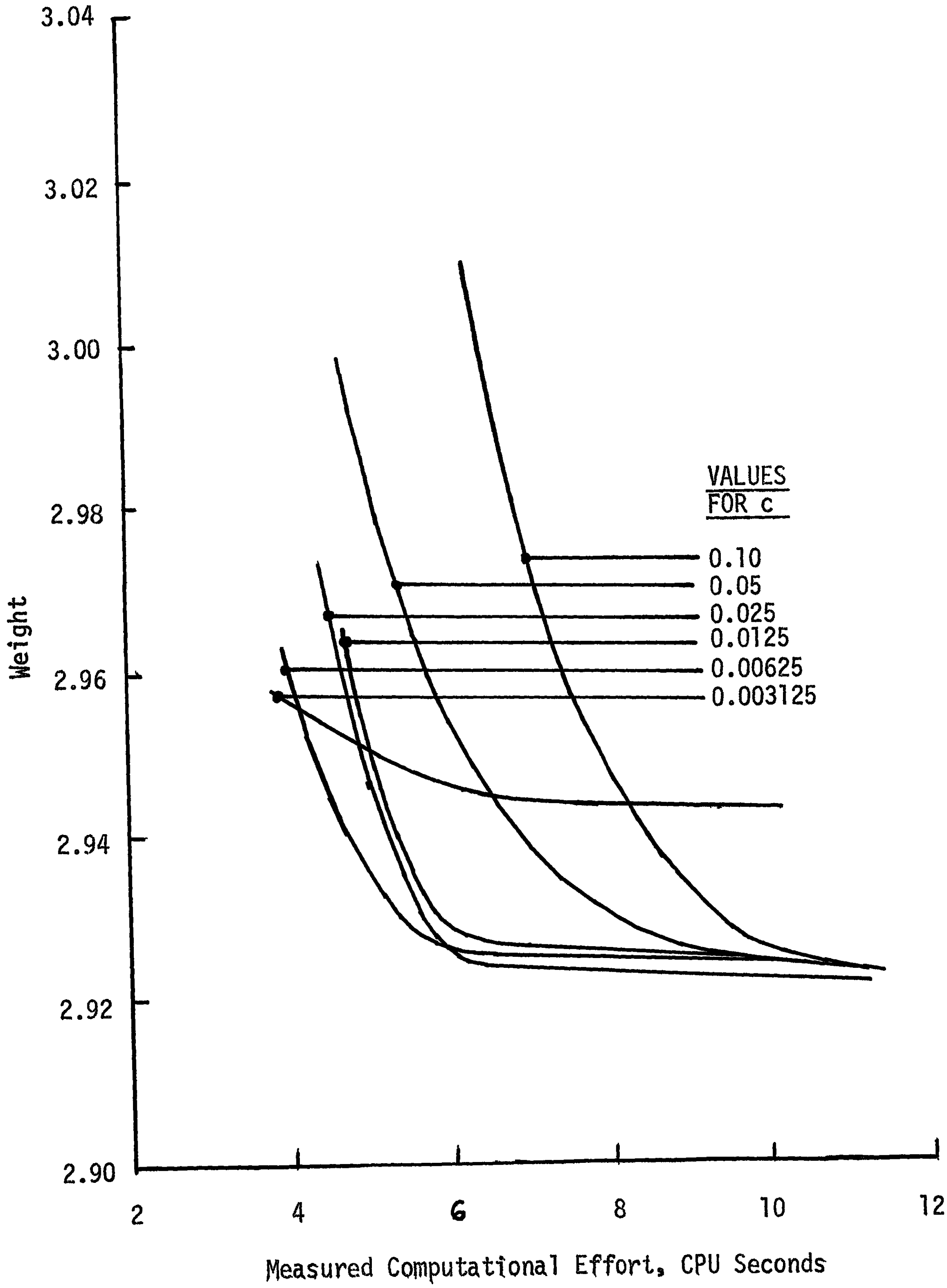

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125
2
4
6
8
10
12
Measured Computational Effort, CPU Seconds

FIGURE 6.4.14

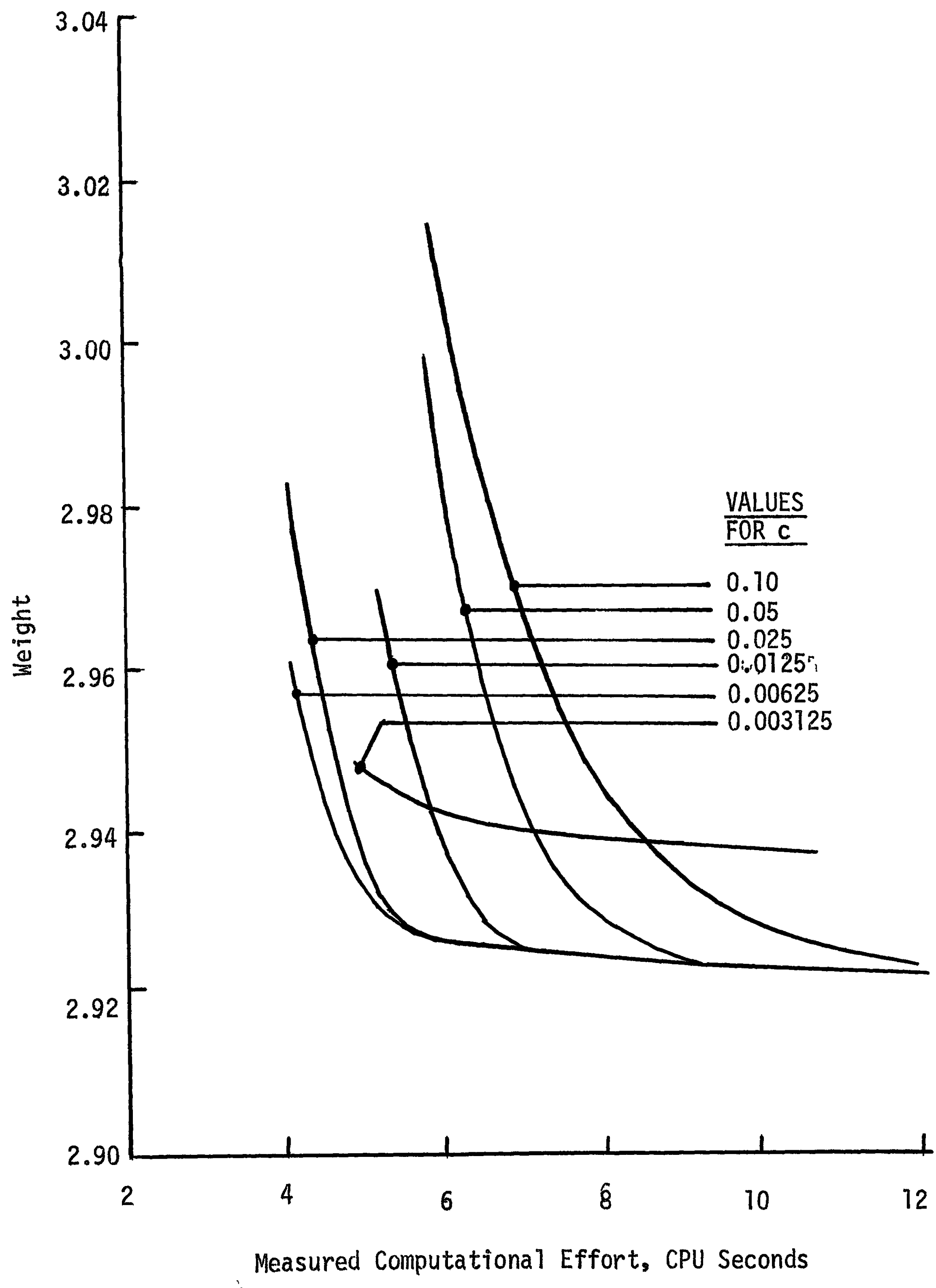

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125
2
4
6
8
10
12
Measured Computational Effort, CPU Seconds

FIGURE 6.4.15

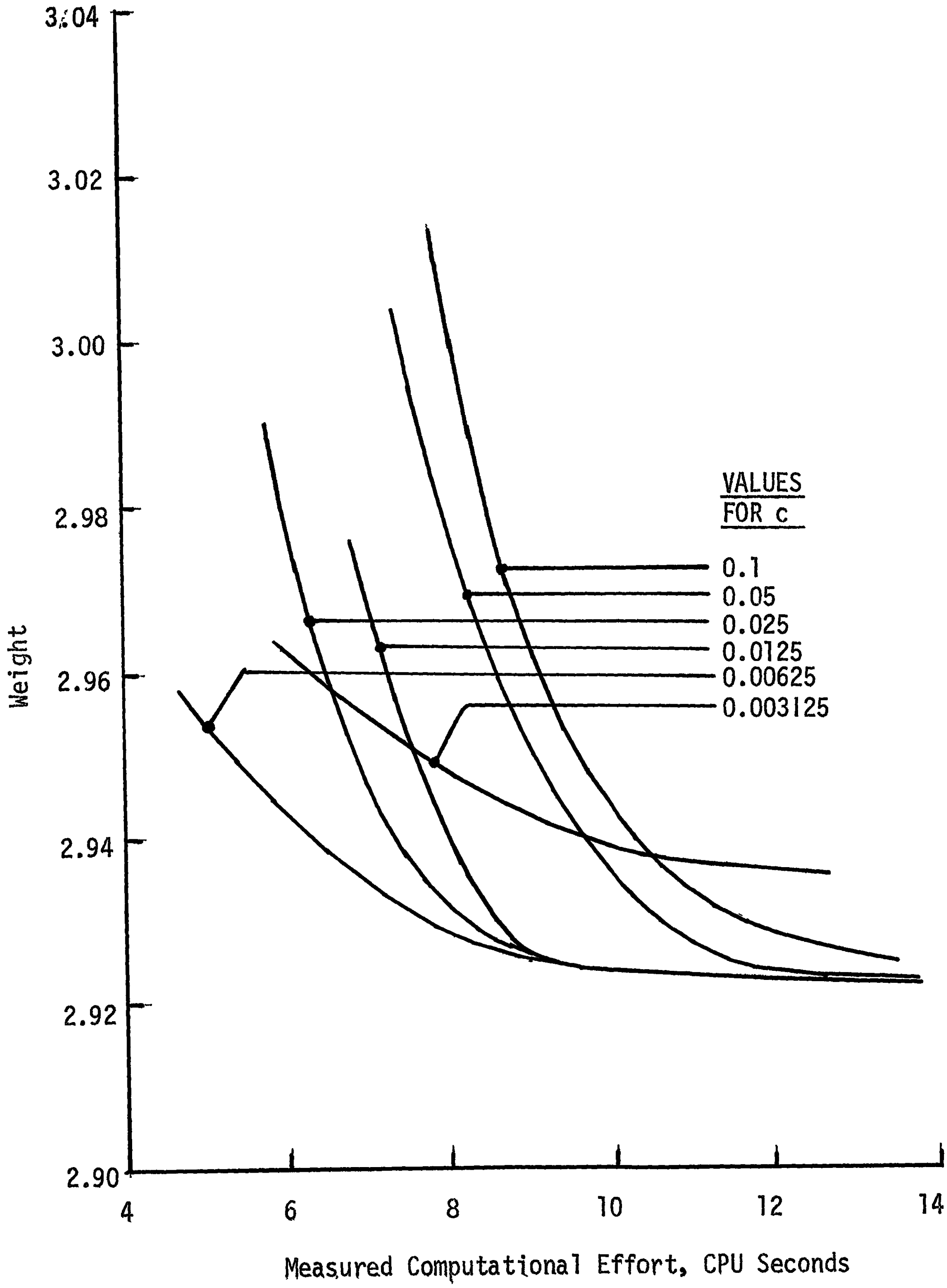

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES
FOR c
0.1
0.05
0.025
0.0125
0.00625
0.003125
4
6
8
10
12
14
Measured Computational Effort, CPU Seconds

FIGURE 6.4.16

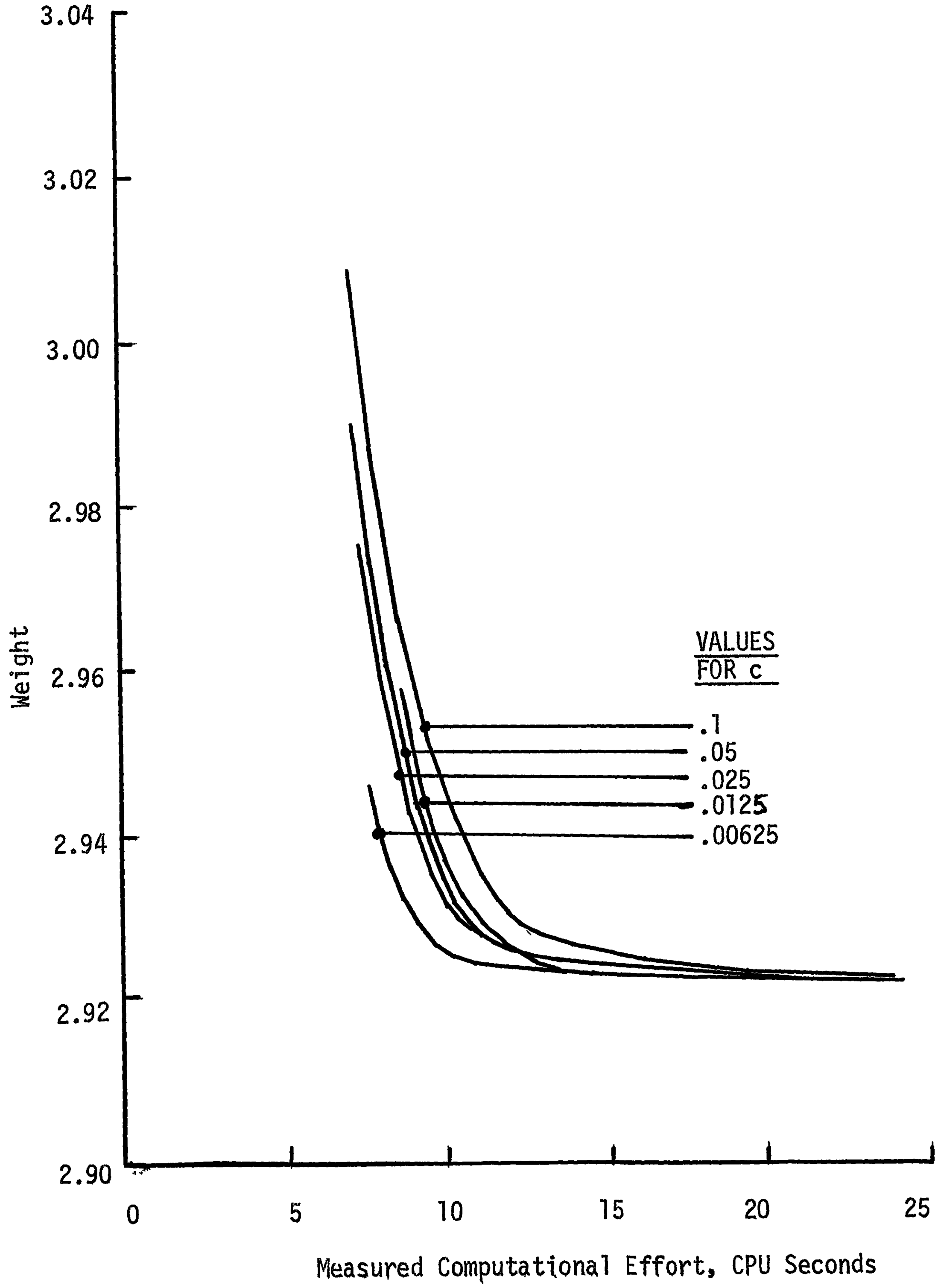

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
.1
.05
.025
.0125
.00625
0
5
10
15
20
25
Measured Computational Effort, CPU Seconds

FIGURE 6.4.17

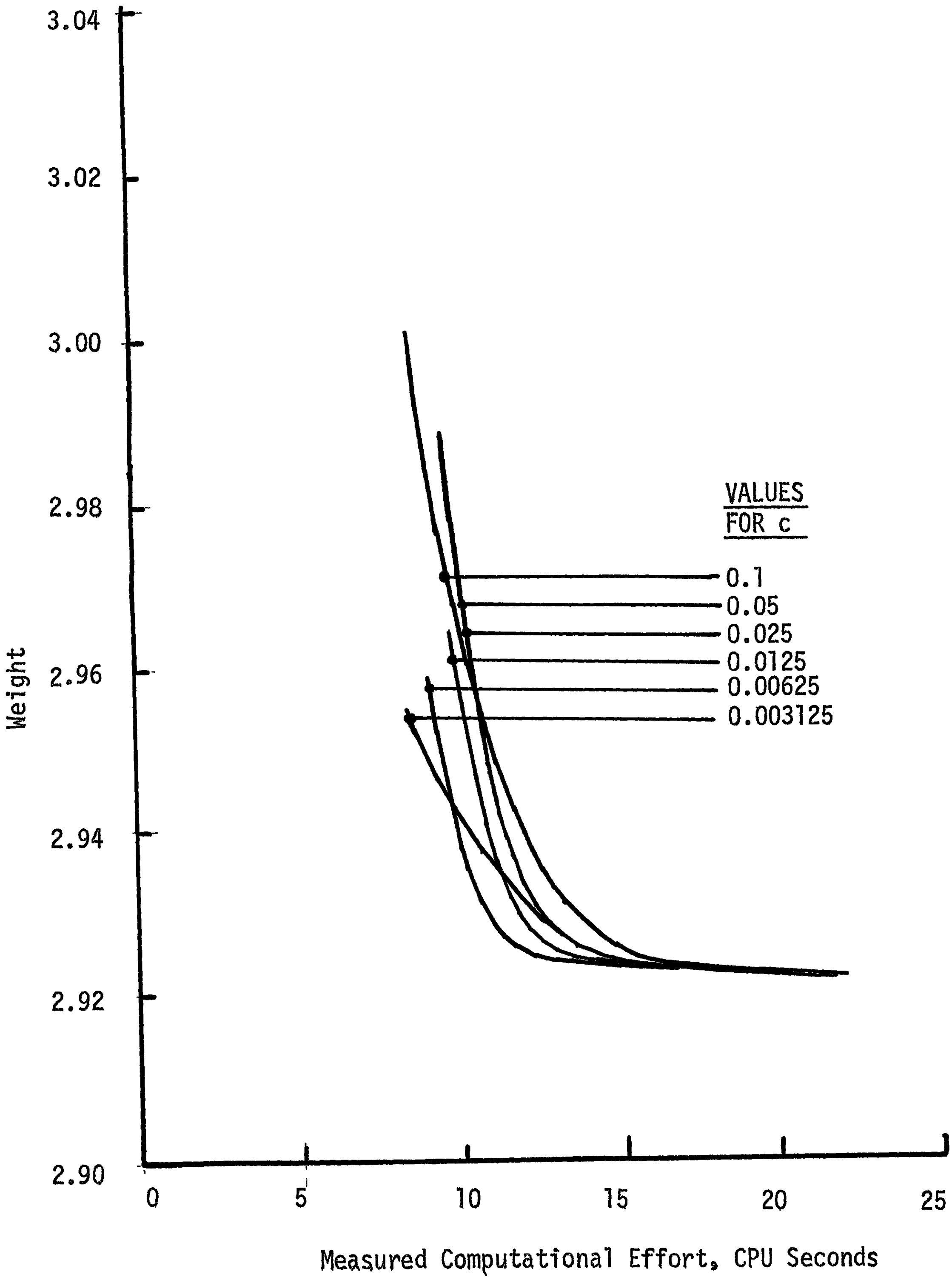

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
0
5
10
15
20
25
Measured Computational Effort, CPU Seconds
VALUES FOR c
0.1
0.05
0.025
0.0125
0.00625
0.003125

FIGURE 6.4.18

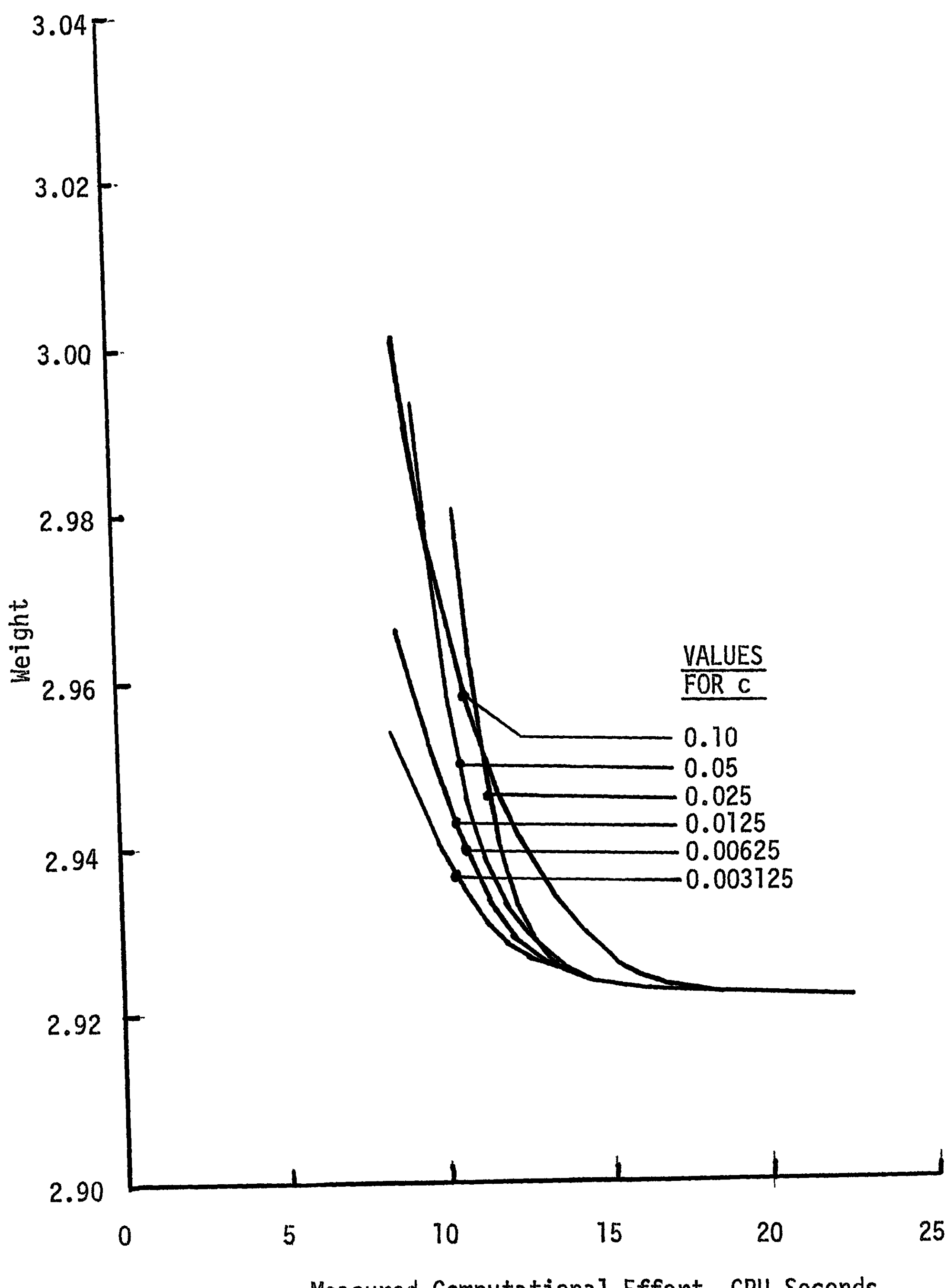

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
0
5
10
15
20
25
Measured Computational Effort, CPU Seconds
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125

FIGURE 6.4.19

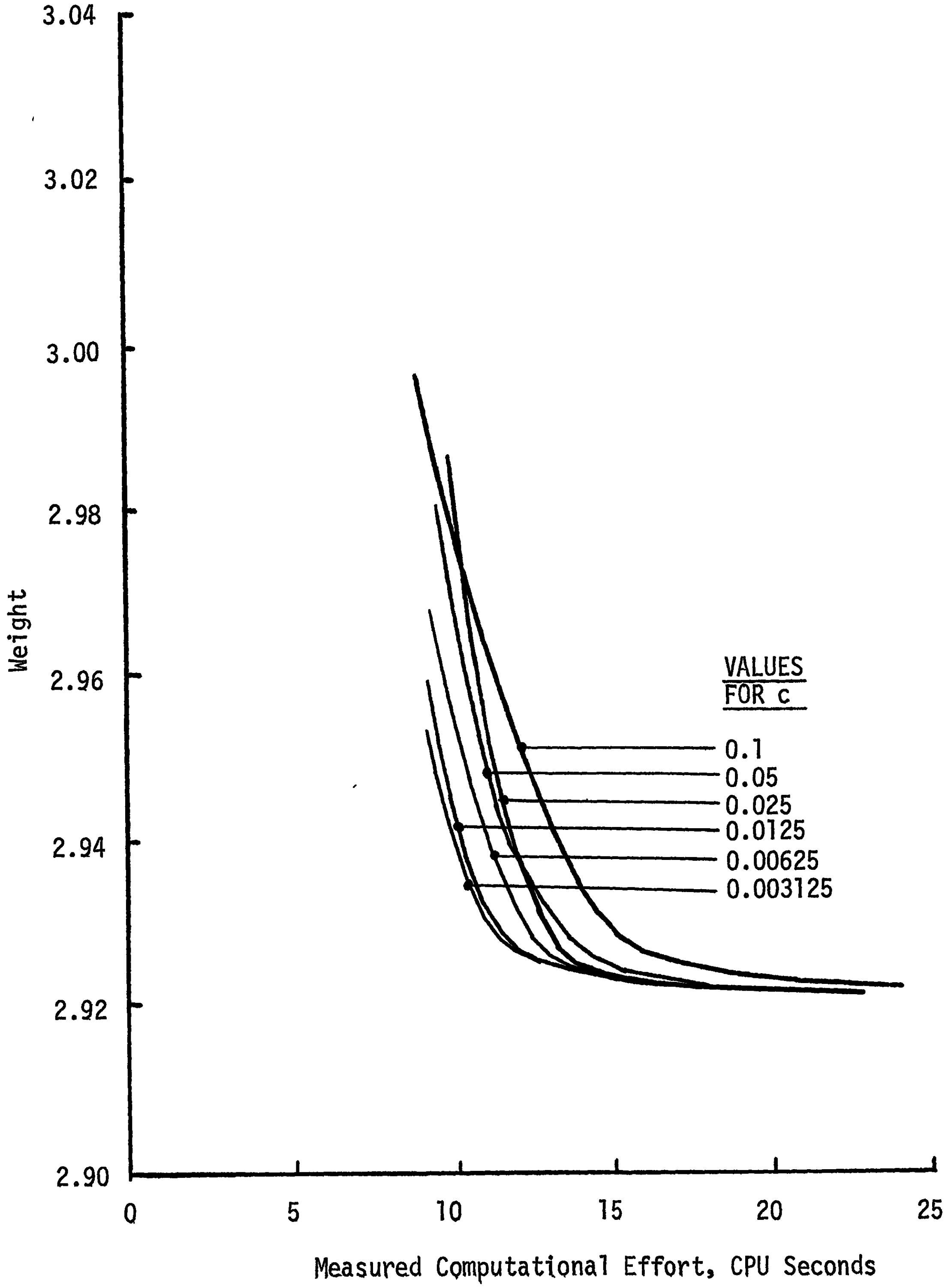
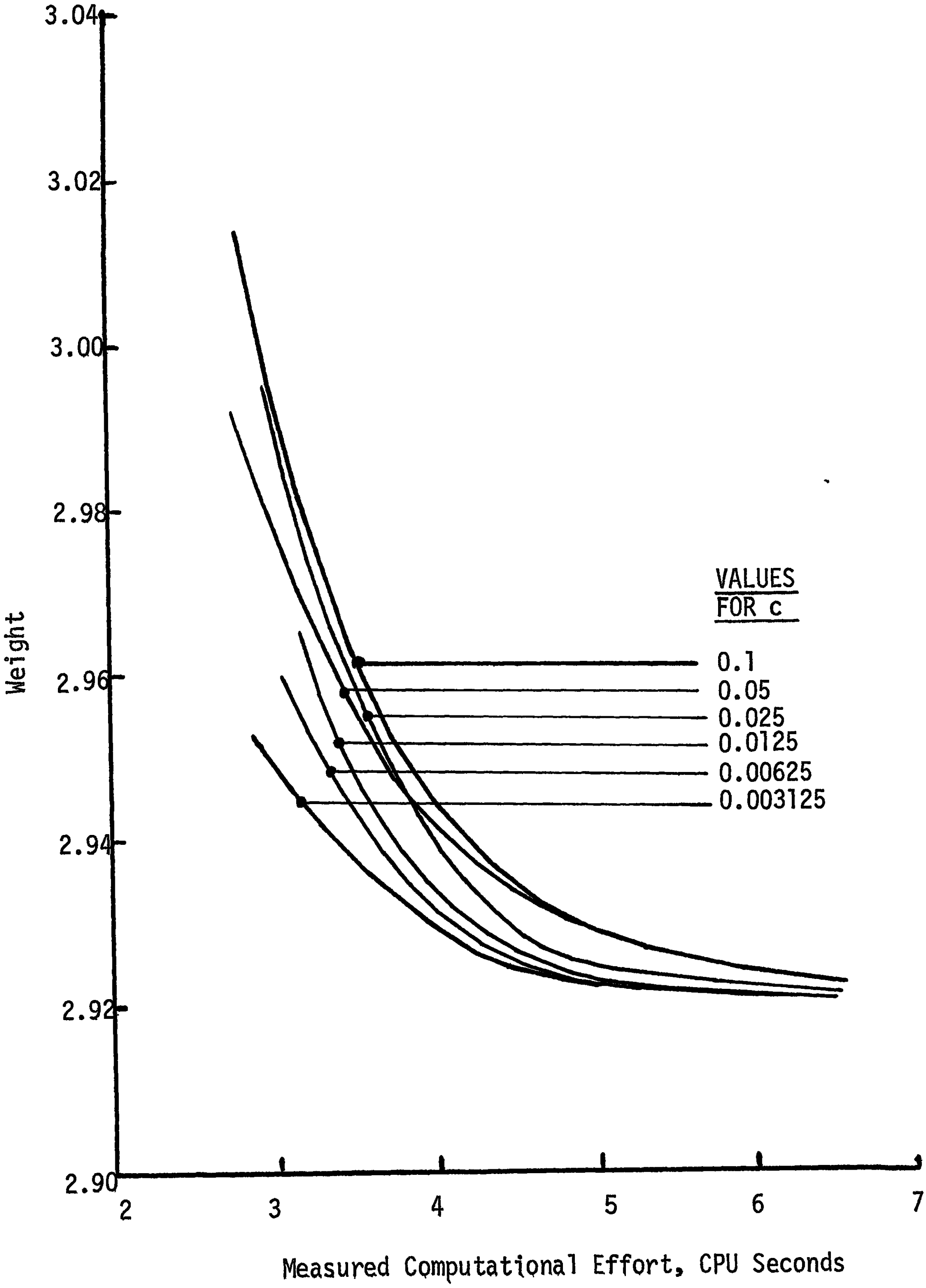

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.1
0.05
0.025
0.0125
0.00625
0.003125
0
5
10
15
20
25
Measured Computational Effort, CPU Seconds

FIGURE 6.4.20



3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
2
3
4
5
6
7
Measured Computational Effort, CPU Seconds
VALUES FOR c
0.1
0.05
0.025
0.0125
0.00625
0.003125

FIGURE 6.4.21

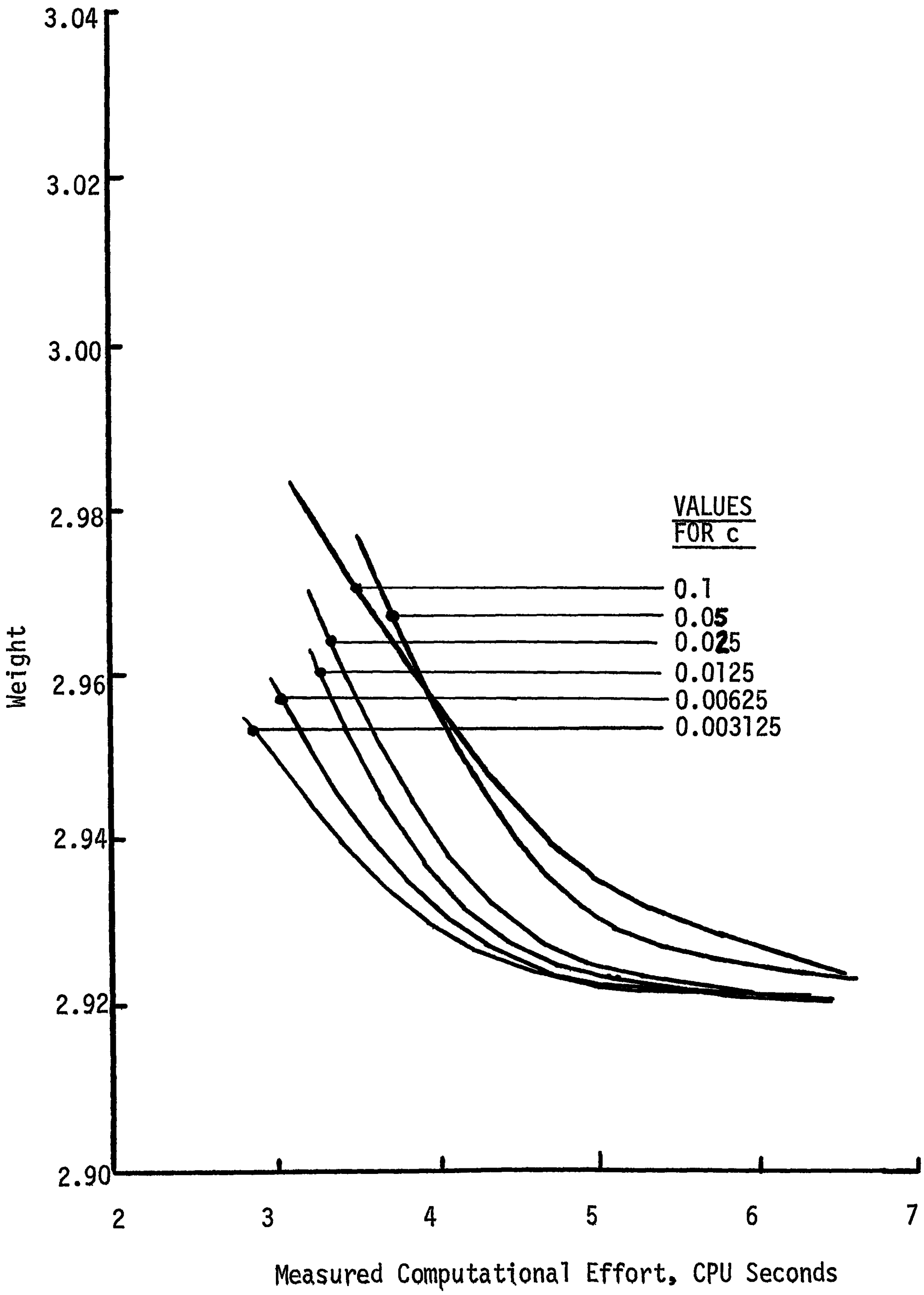

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.1
0.05
0.025
0.0125
0.00625
0.003125
2
3
4
5
6
7
Measured Computational Effort, CPU Seconds

FIGURE 6.4.22

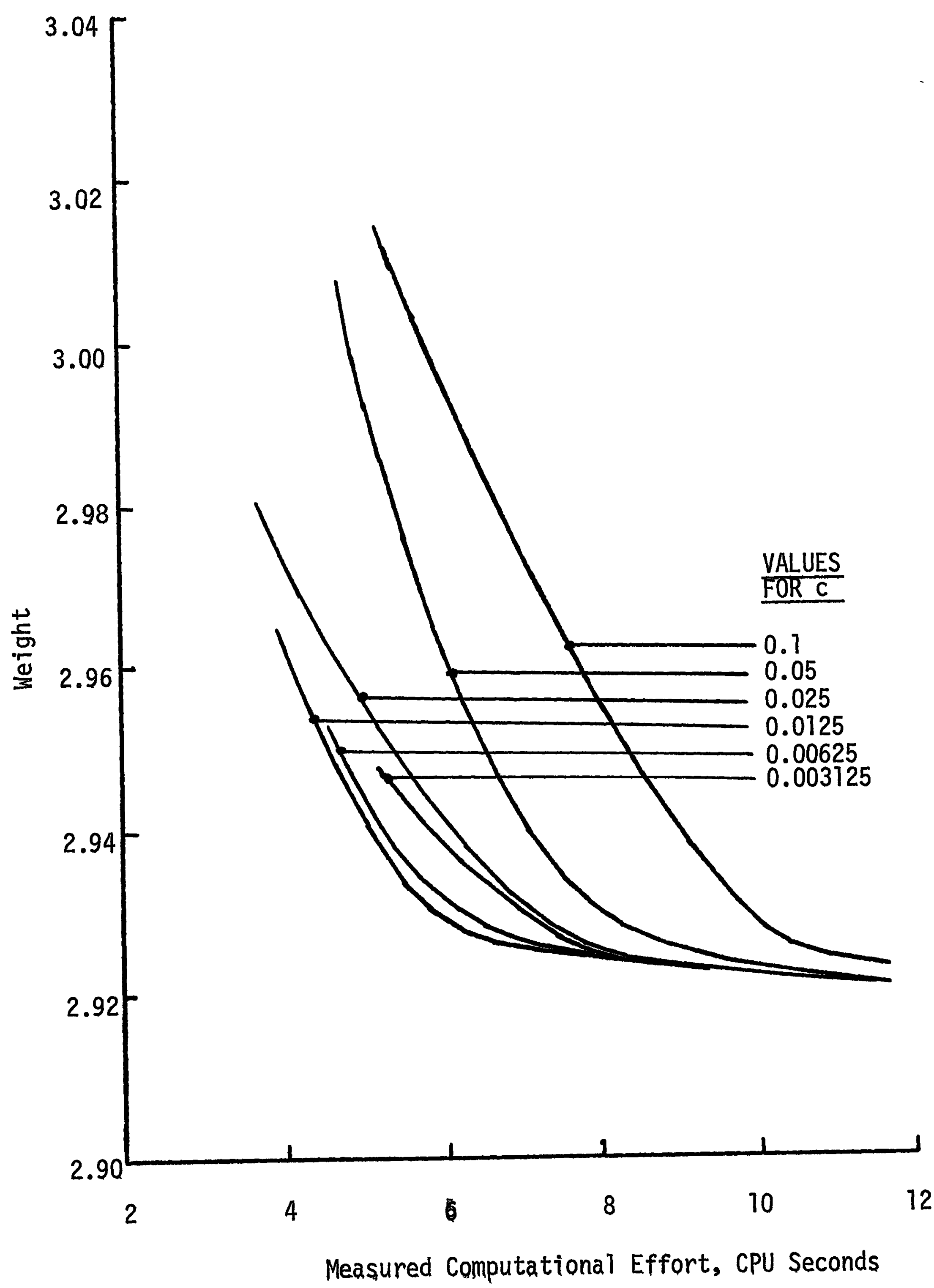

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.1
0.05
0.025
0.0125
0.00625
0.003125
2
4
6
8
10
12
Measured Computational Effort, CPU Seconds

FIGURE 6.4.23

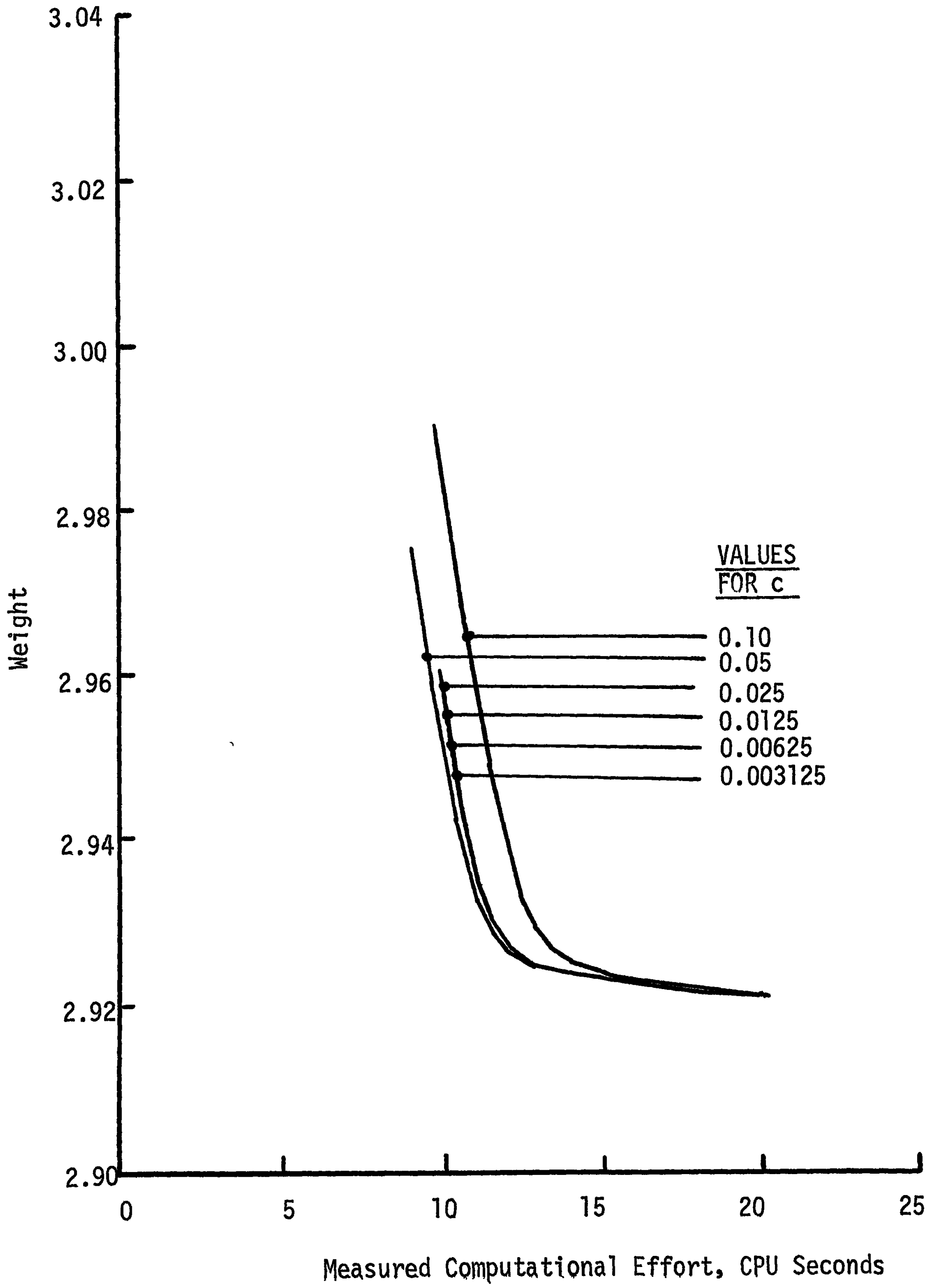

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
VALUES FOR c
0.10
0.05
0.025
0.0125
0.00625
0.003125
0
5
10
15
20
25
Measured Computational Effort, CPU Seconds

FIGURE 6.4.24

CHAPTER 7

TEST RESULTS AND DISCUSSION

7.1 Introduction

Throughout this chapter the following definitions are used:

computational effort = CPU time expended using an IBM 370/145 computer,

A-B effort ratio = $E_{A,B}$ = computational effort either to perform operation A or using method A divided by computational effort either to perform operation B or using method B,

minimum weight (MW) = lowest recorded weight of all feasible designs encountered by any of the algorithms

near minimum weight (NMW) = 100.5% of the minimum weight defined above.

Slight changes in the arbitrary coefficients can alter the computational effort required by the algorithms. Nevertheless it is assumed in chapter 6 that either the optimum choice or an equally non-optimum choice has been made for the arbitrary coefficients of the algorithms on all of the problems.

7.2 Computer results.

Table 7.2.1 shows on which problems the algorithms were tested and indicates whether the near minimum was achieved.

Failure to achieve the near minimum weight (NMW) was usually caused by the upper time limits set for the computer run. However, MFD failed to achieve the near minimum on both the 21 bar truss and the 25 node plate because the LP algorithm lacked adequate precautions to prevent cycling. Powell's method and Stewart's method were not run on the large problems because earlier runs on the smaller problems had established that these algorithms were not as efficient as the other algorithms.

Table 7.2.1: Computer tests made

	P-BAR TRUSSES				P-NODE PLATES			
ALG	3	7	13	21	4	9	16	25
MAP	Y	Y	Y	Y	Y	Y	Y	Y
MFD	Y	Y	Y	N	Y	Y	Y	N
N1	Y	Y	Y	Y	Y	Y	Y	Y
N2	Y	Y	Y	N	Y	Y	Y	Y
FP	Y	Y	Y	Y	Y	Y	Y	Y
ST	Y	N	N	-	Y	Y	Y	-
PO	Y	N	-	-	Y	Y	-	-

Y = algorithm reaches the NMW of the problem,

N = algorithm does not find the NMW within the time allowed, and

\- = problem not run using this algorithm.

Figures 7.2.2 to 7.2.9 plot weight against total computational effort used during the computer runs shown in table 7.2.1.

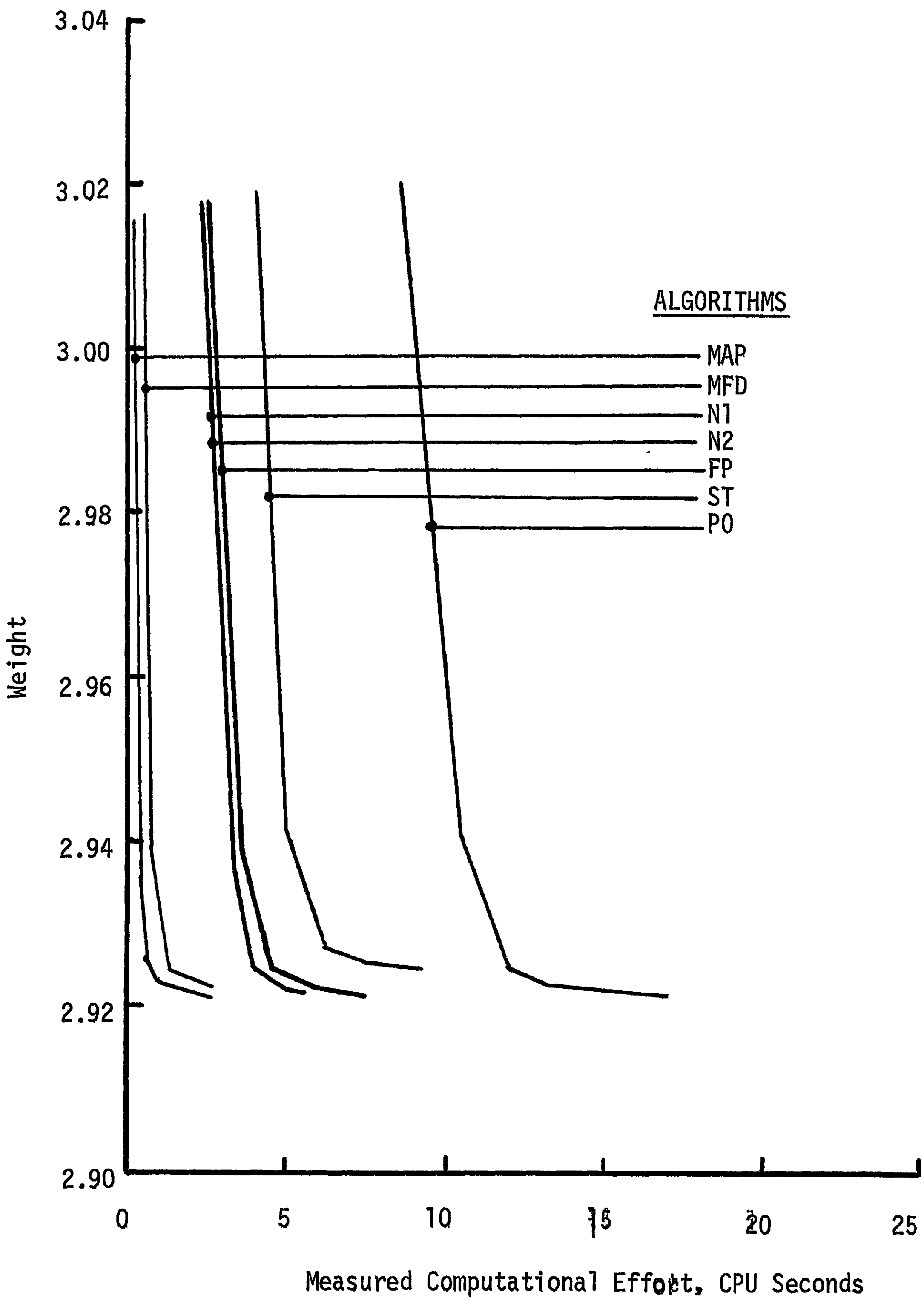

3.04
3.02
3.00
2.98
2.96
2.94
2.92
2.90
Weight
ALGORITHMS
MAP
MFD
N1
N2
FP
ST
PO
0
5
10
15
20
25
Measured Computational Effort, CPU Seconds

FIGURE 7.2.2

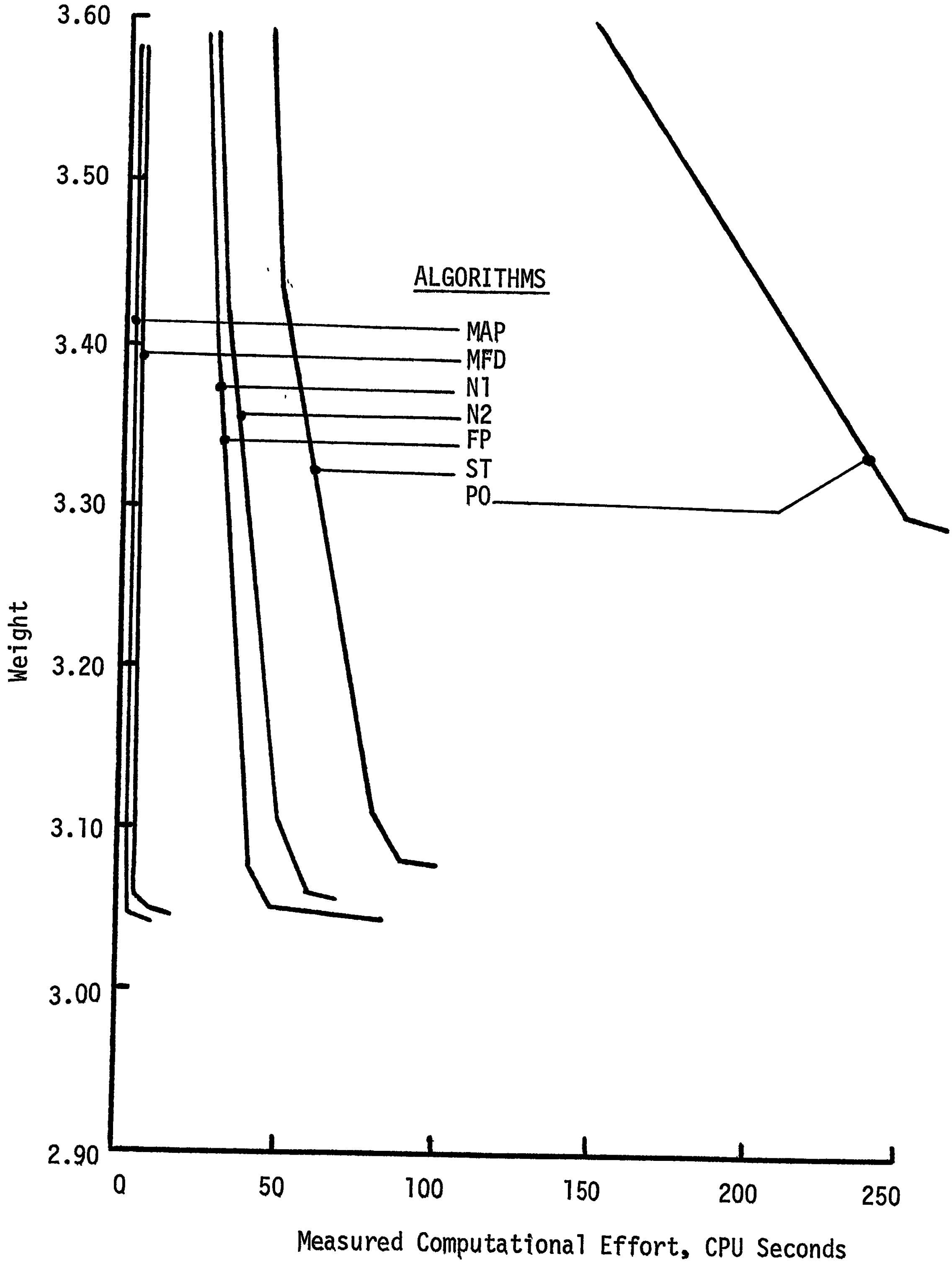

3.60
3.50
3.40
3.30
3.20
3.10
3.00
2.90
Weight
ALGORITHMS
MAP
MFD
N1
N2
FP
ST
PO
0
50
100
150
200
250
Measured Computational Effort, CPU Seconds

FIGURE 7.2.3

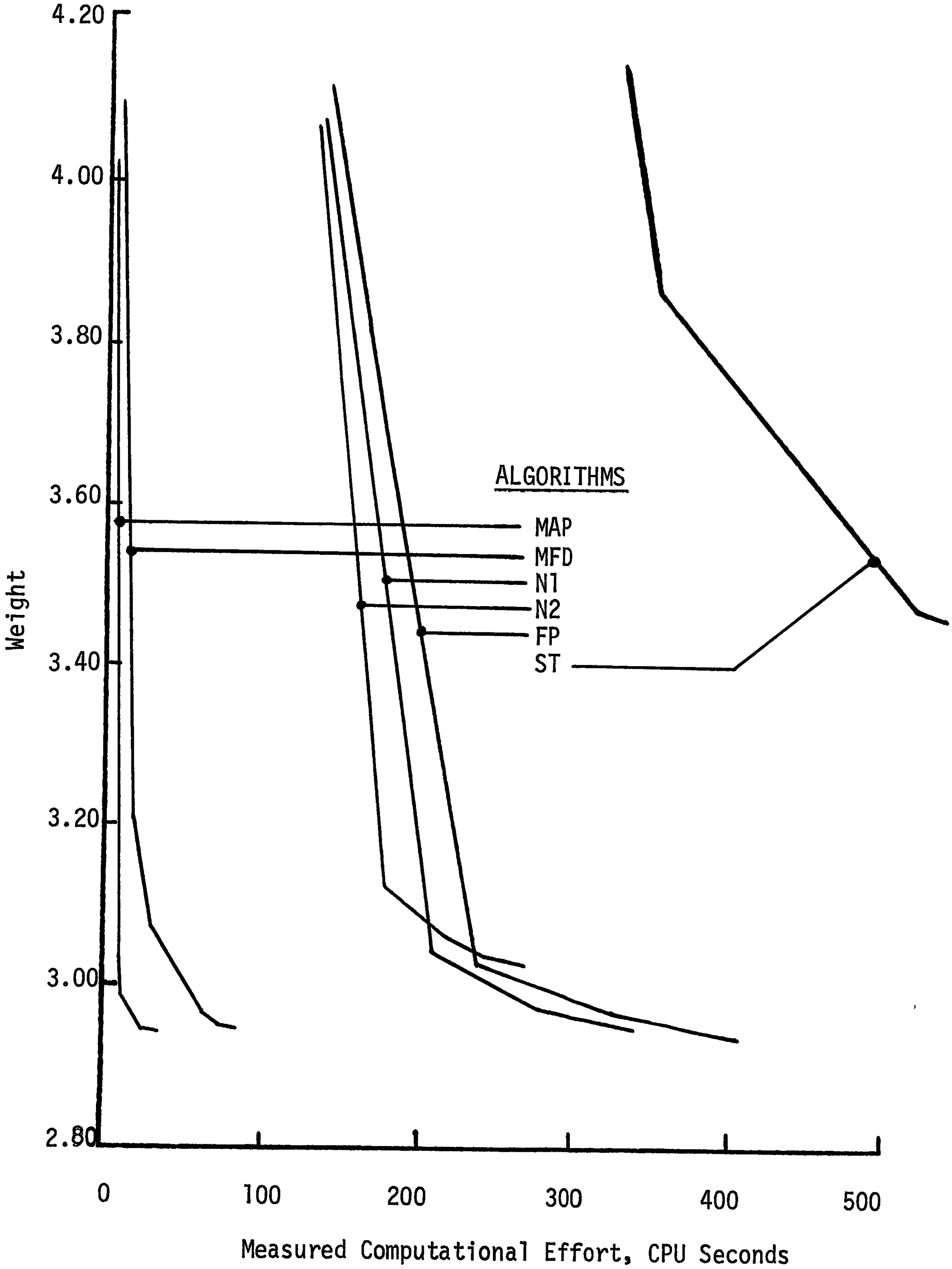

4.20
4.00
3.80
3.60
3.40
3.20
3.00
2.80
Weight
ALGORITHMS
MAP
MFD
N1
N2
FP
ST
0
100
200
300
400
500
Measured Computational Effort, CPU Seconds

FIGURE 7.2.4

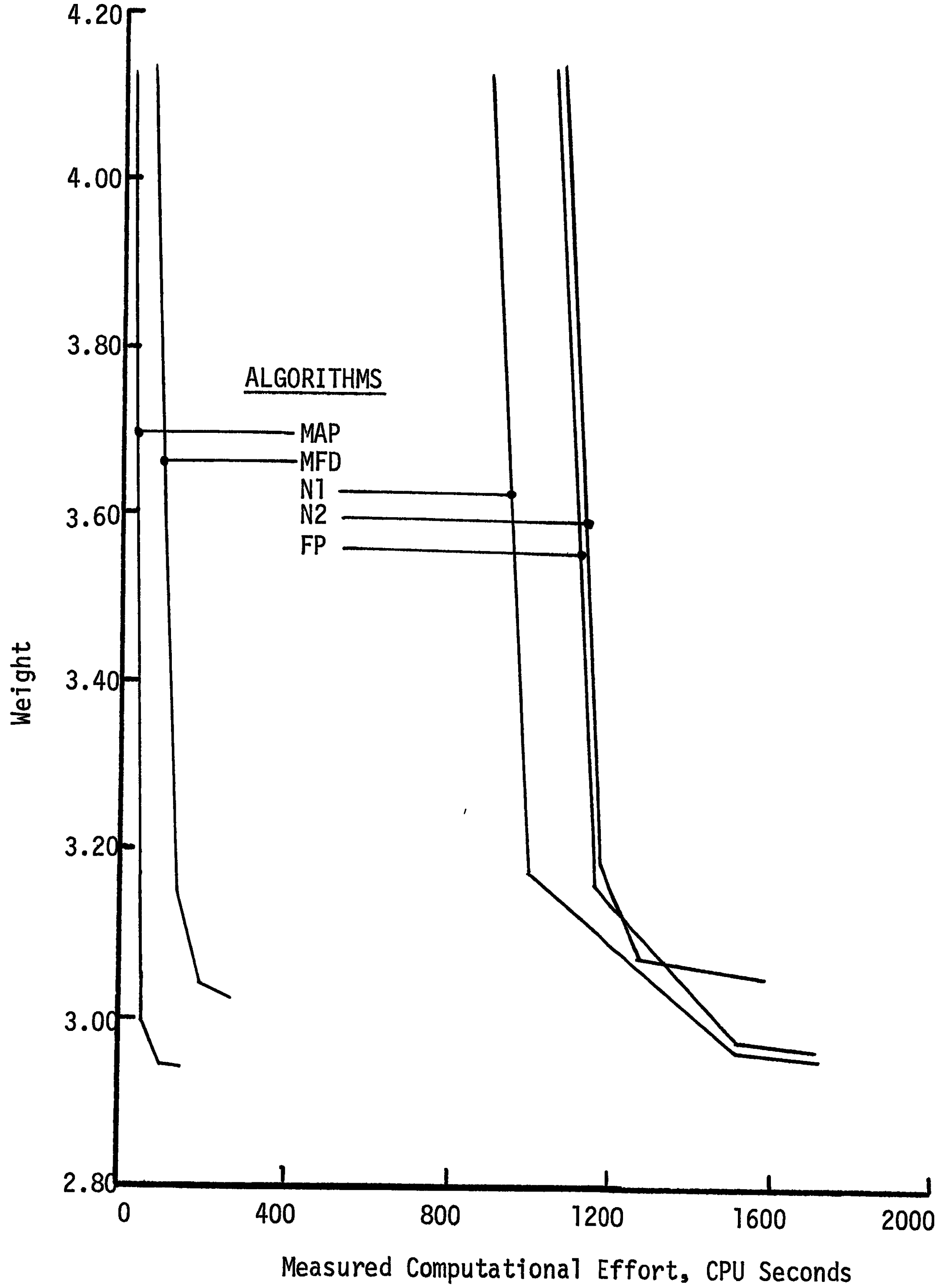

4.20
4.00
3.80
3.60
3.40
3.20
3.00
2.80
Weight
ALGORITHMS
MAP
MFD
N1
N2
FP
0
400
800
1200
1600
2000
Measured Computational Effort, CPU Seconds

FIGURE 7.2.5

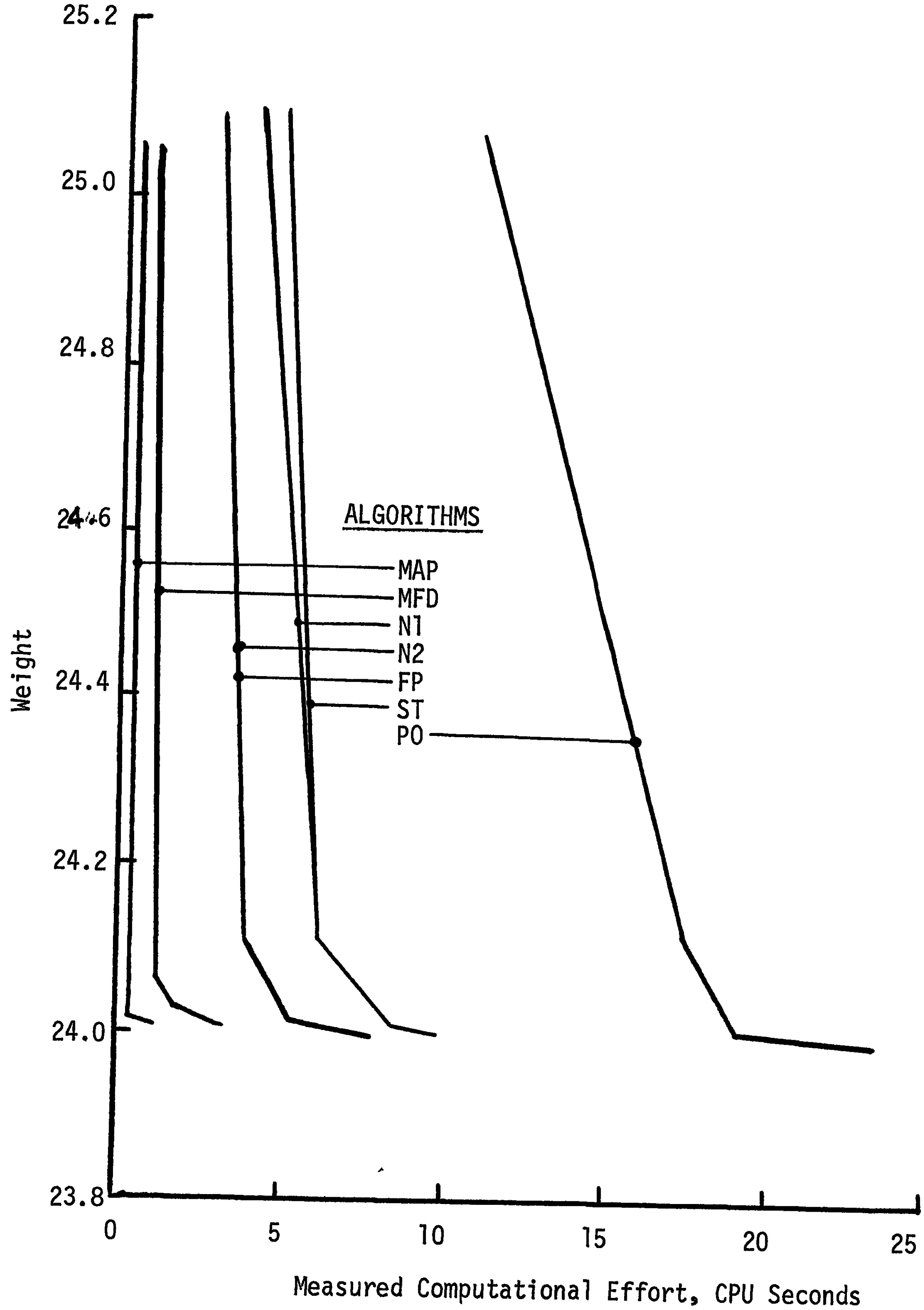

25.2
25.0
24.8
24.6
24.4
24.2
24.0
23.8
Weight
ALGORITHMS
MAP
MFD
N1
N2
FP
ST
PO
0
5
10
15
20
25
Measured Computational Effort, CPU Seconds

FIGURE 7.2.6

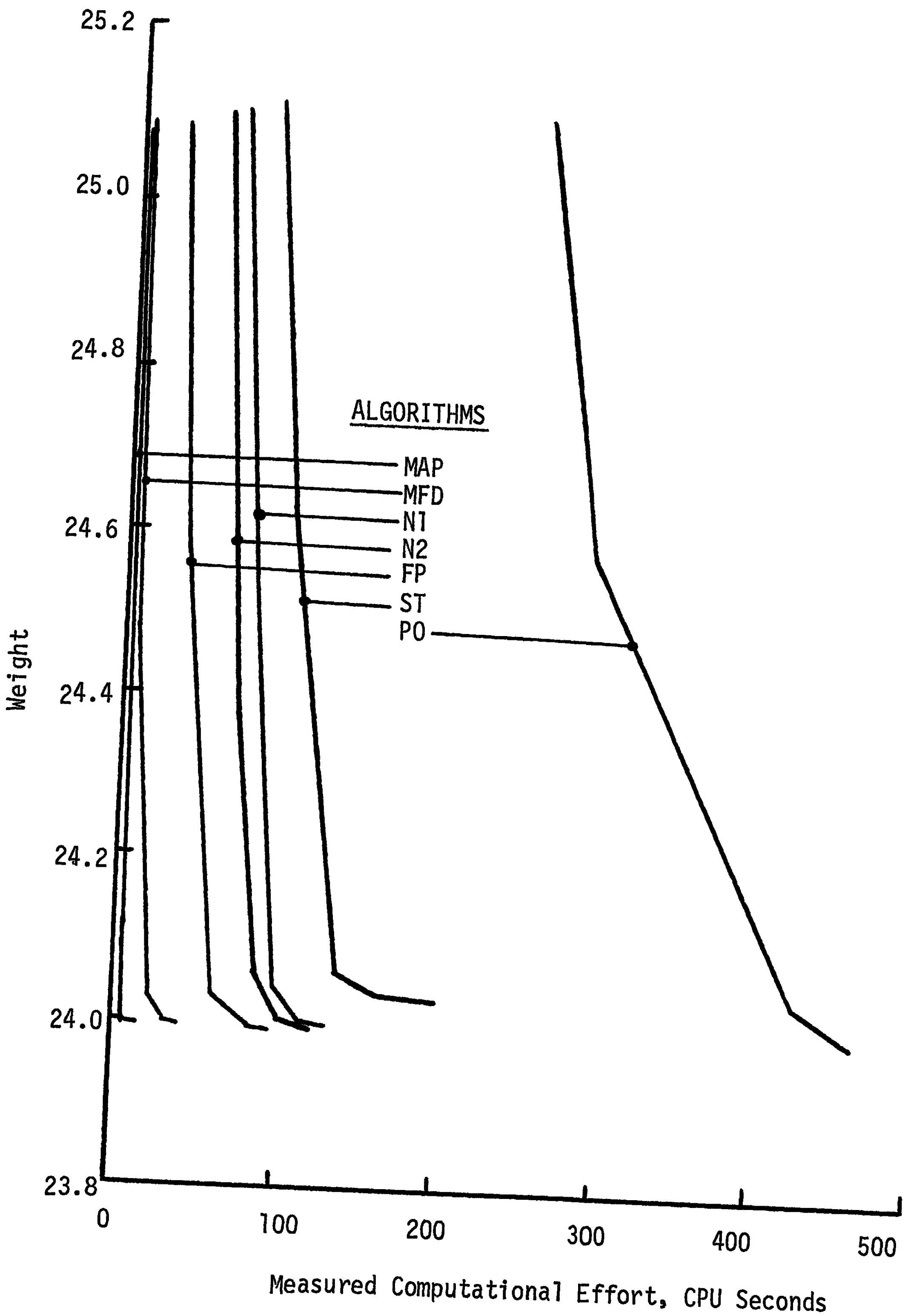
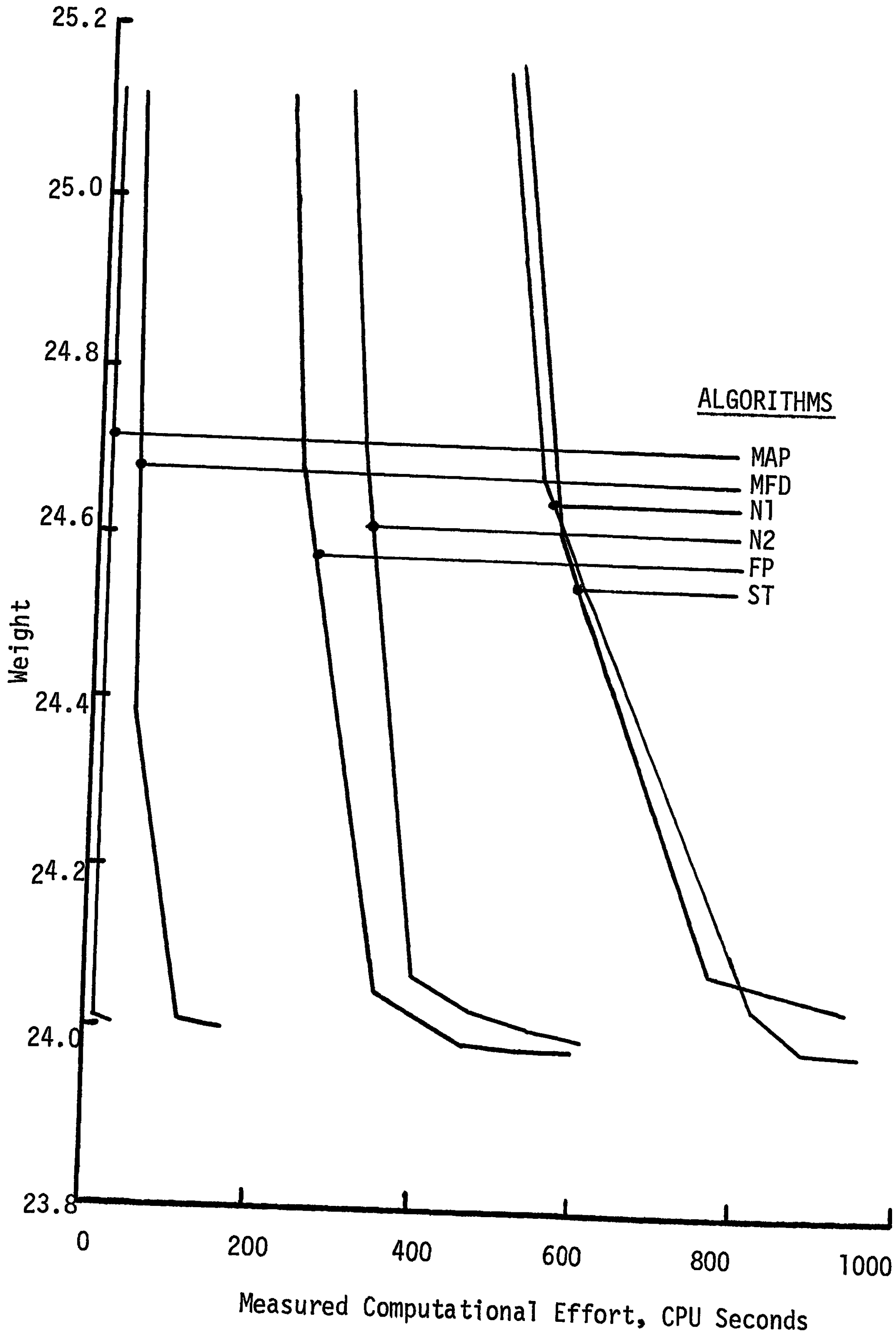

25.2
25.0
24.8
24.6
24.4
24.2
24.0
23.8
Weight
ALGORITHMS
MAP
MFD
N1
N2
FP
ST
PO
0
100
200
300
400
500
Measured Computational Effort, CPU Seconds

FIGURE 7.2.7



25.2
25.0
24.8
24.6
24.4
24.2
24.0
23.8
Weight
ALGORITHMS
MAP
MFD
N1
N2
FP
ST
0
200
400
600
800
1000
Measured Computational Effort, CPU Seconds

FIGURE 7.2.8

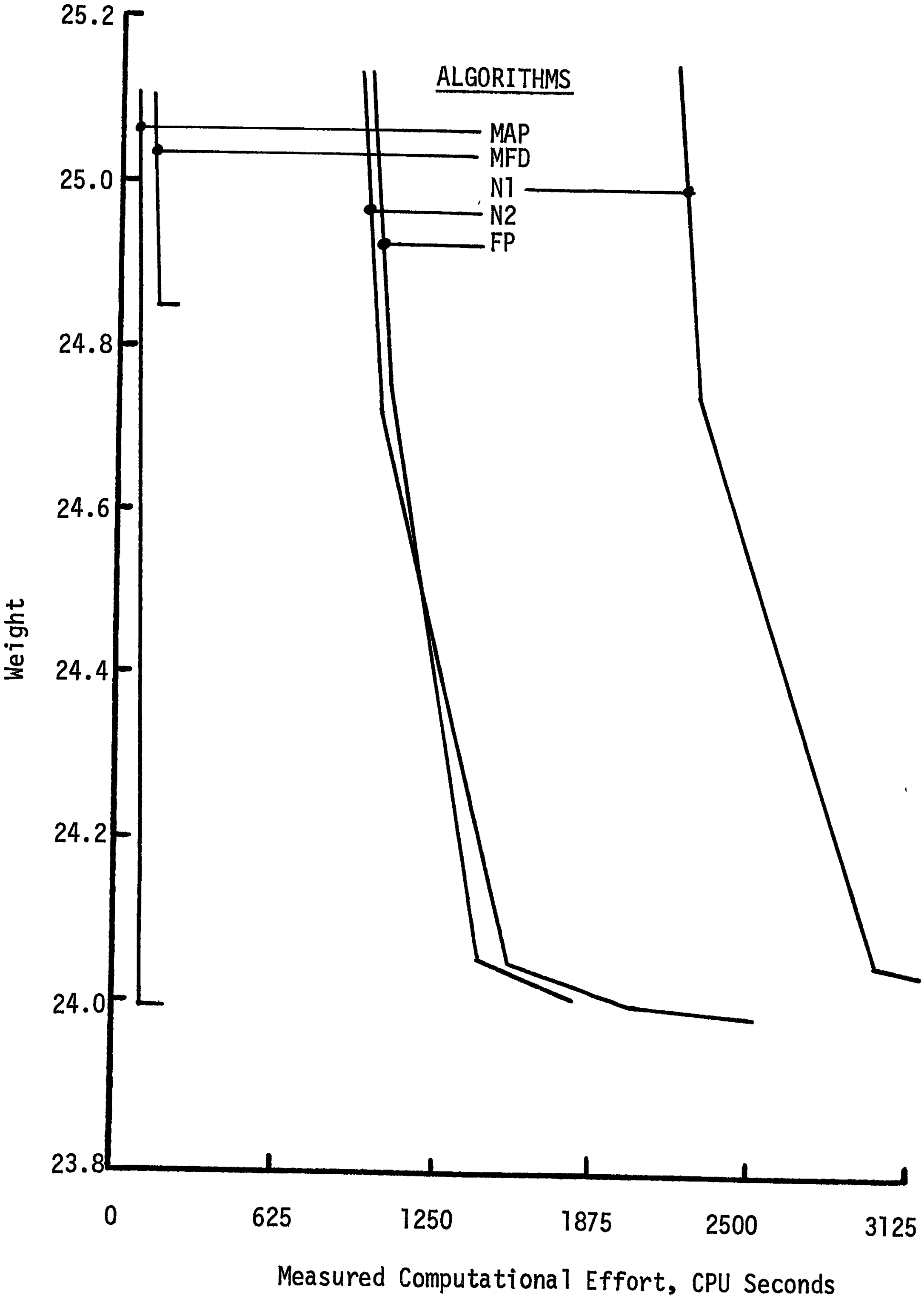

ALGORITHMS
MAP
MFD
N1
N2
FP
25.2
25.0
24.8
24.6
24.4
24.2
24.0
23.8
Weight
0
625
1250
1875
2500
3125
Measured Computational Effort, CPU Seconds

FIGURE 7.2.9

Table 7.2.10 shows values of the parameters measured at the NMW for each run. The parameters measured and the abbreviations used are:

TF = CPU time (secs) used in evaluating functions,

TD = CPU time (secs) used in evaluating derivatives,

TO = CPU time (secs) used in the optimisation algorithms,

TT = sum of TF, TD and TO,

NFE = total number of functions evaluations,

NDE = total number of derivative evaluations, and

NITS = total number of iterations.

The results presented in the remainder of this chapter have been derived from the values in table 7.2.10.

7.3 Effort used by the function and derivative algorithms

For the trusses shown in figures 6.2.1 and for the plates shown in figure 6.2.3, figures 7.3.1 and 7.3.2 respectively, plot measured computational effort for evaluation of functions and derivatives against the number of design variables. Table 7.3.3 gives the derivative - function effort ratios, A/B, obtained from the results shown in figures 7.3.1 and 7.3.2.

The effort ratios for rows 1 and 2 in table 7.3.3 are for derivatives obtained by differentiation, and the ratios in rows 3 and 4 are for derivatives obtained from a forward finite difference scheme. Note that differentiation - finite difference effort ratios for first derivatives shown in row 5 are of a similar magnitude to the second derivative ratios shown in row 6.

Table 7.2.10: Results measured at the NMW.

ALG		P-BAR TRUSSES, P =				P-NODE PLATES, P =			
		3	7	13	21	4	9	16	25
MAP	TF	.0088	0.54	4.2	14.2	.107	0.77	2.2	5.
	TD	.115	1.40	16.4	65.4	.234	3.80	17.0	56
	TO	.071	0.41	5.4	19.1	.141	1.45	9.4	40.
	TT	.274	2.35	26.0	98.7	.482	6.02	28.6	101.
	NFE	3	5	11	12	3	5	5	5
	NDE	2	4	10	11	2	4	4	4
	NITS	2	4	10	11	2	4	4	4
MFD	TF	.352	3.55	52.	-	.249	2.1	10.	-
	TD	.369	4.04	43.	-	.466	8.9	78.	-
	TO	.270	1.71	15.	-	.202	1.5	17.	-
	TT	.991	9.30	110.	-	.917	12.5	105.	-
	NFE	11	32	138	-	7	13	22	-
	NDE	6	11	26	-	4	9	18	-
	NITS	6	11	26	-	4	9	18	-
N1	TF	1.57	14.3	57.	196.	2.25	11.0	42.0	93
	TD	1.87	32.2	251.	1320	3.66	84.2	776.	4300
	TO	0.16	0.6	2.	15	0.34	0.9	2.	5
	TT	3.60	47.1	310.	1530	6.25	96.1	820.	4398
	NFE	51	133	157	172	63	70	93	91
	NDE	9	17	19	20	7	12	15	14
	NITS	9	17	19	20	7	12	15	14
FP	TF	2.71	28.6	184.	801.	2.16	28.7	116.	397.
	TD	0.92	16.9	144.	706.	1.48	30.6	221.	1067
	TO	0.26	1.5	6.	15.	0.36	1.5	4.	10
	TT	3.89	47.0	344.	1522	4.00	60.8	341.	1474
	NFE	87	259	485	704	60	179	256	382
	NDE	14	42	84	116	12	30	49	74
	NITS	13	43	85	117	11	30	49	74
ST	TF	5.15	-	-	-	5.75	137.	761.	-
	TD	-	-	-	-	-	-	-	-
	TO	0.36	-	-	-	0.50	2.	6.	-
	TT	5.51	-	-	-	6.25	140.	767.	-
	NFE	162	-	-	-	156	880	1690	-
	NDE	-	-	-	-	-	-	-	-
	NITS	15	-	-	-	12	45	57	-
PQ	TF	12.1	-	-	-	16.6	417.	-	-
	TD	-	-	-	-	-	-	-	-
	TO	0.8	-	-	-	1.1	7.	-	-
	TT	12.9	-	-	-	17.7	424.	-	-
	NFE	387	-	-	-	470	2679	-	-
	NDE	-	-	-	-	-	-	-	-
	NITS	17.3	-	-	-	17.0	47.2	-	-

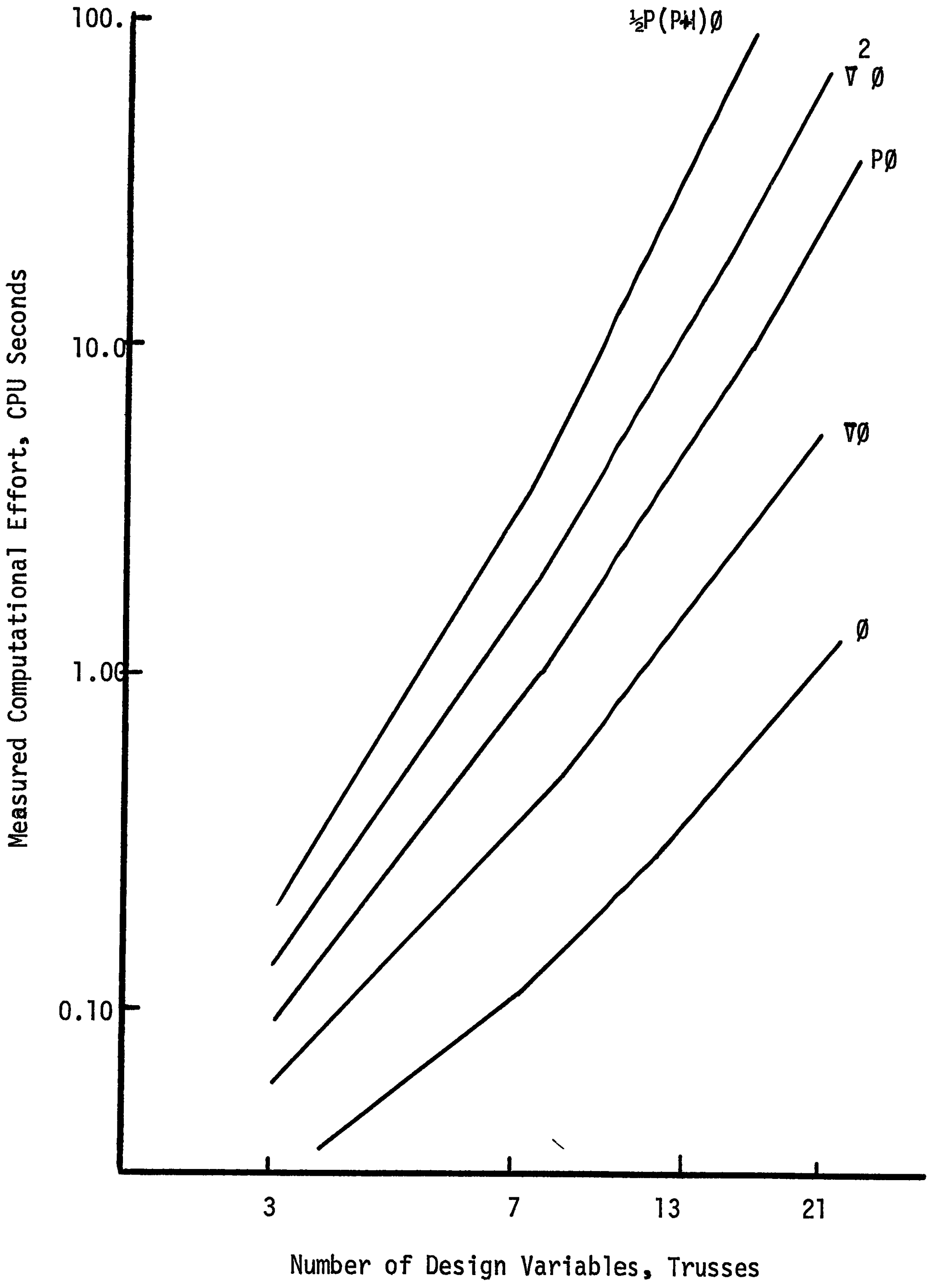
In the figures 7.3.1 and 7.3.2 and in the table 7.3.3, it can be seen that derivatives obtained by differentiation required in general less computational effort than derivatives obtained by finite differencing. The values in table 7.3.3 compare well with the estimates given in table 5.4.30.

Table 7.3.3: Measured function and derivative effort ratios, $E_{A,B}$.

A	B	P-BAR TRUSSES, P = 3	7	13	21	P-NODE PLATES, P = 4	9	16	25
$\nabla\phi$	ϕ	1.98	3.34	4.43	5.12	3.34	6.35	9.81	13.7
$\nabla^2\phi$	ϕ	4.31	14.4	31.3	55.2	10.1	39.4	107.	196.
$P\times\phi$	ϕ	3.00	7.00	13.0	21.0	4.00	9.00	16.0	25.0
$\frac{(P^2+P)\phi}{2}$	ϕ	6.00	28.0	91.0	231.	10.0	45.0	136.	325.
$\nabla\phi$	$P\times\phi$	0.66	0.48	0.34	0.24	0.83	0.70	0.61	0.55
$\nabla^2\phi$	$\frac{(P^2+P)\phi}{2}$	0.72	0.52	0.34	0.24	1.01	0.88	0.79	0.60

7.4 Effort used by the optimisation algorithms.

For the trusses shown in figure 6.2.1 and for the plates shown in figure 6.2.3, figures 7.4.1 and 7.4.2 respectively plot measured computational effort used during one iteration (as defined in chapter 5) of each of the algorithms against the number of design variables. Comparison of figures 7.4.1 and 7.4.2 with figures 5.5.13 and 5.5.14 shows that except when MAP is being considered, the estimated algorithm iteration effort ratios agree with the measured ratios. When considering MAP the discrepancies arising



100.
10.0
1.00
0.10
Measured Computational Effort, CPU Seconds
½P(P+H)Ø
∇²Ø
PØ
∇Ø
Ø
3
7
13
21
Number of Design Variables, Trusses

FIGURE 7.3.1

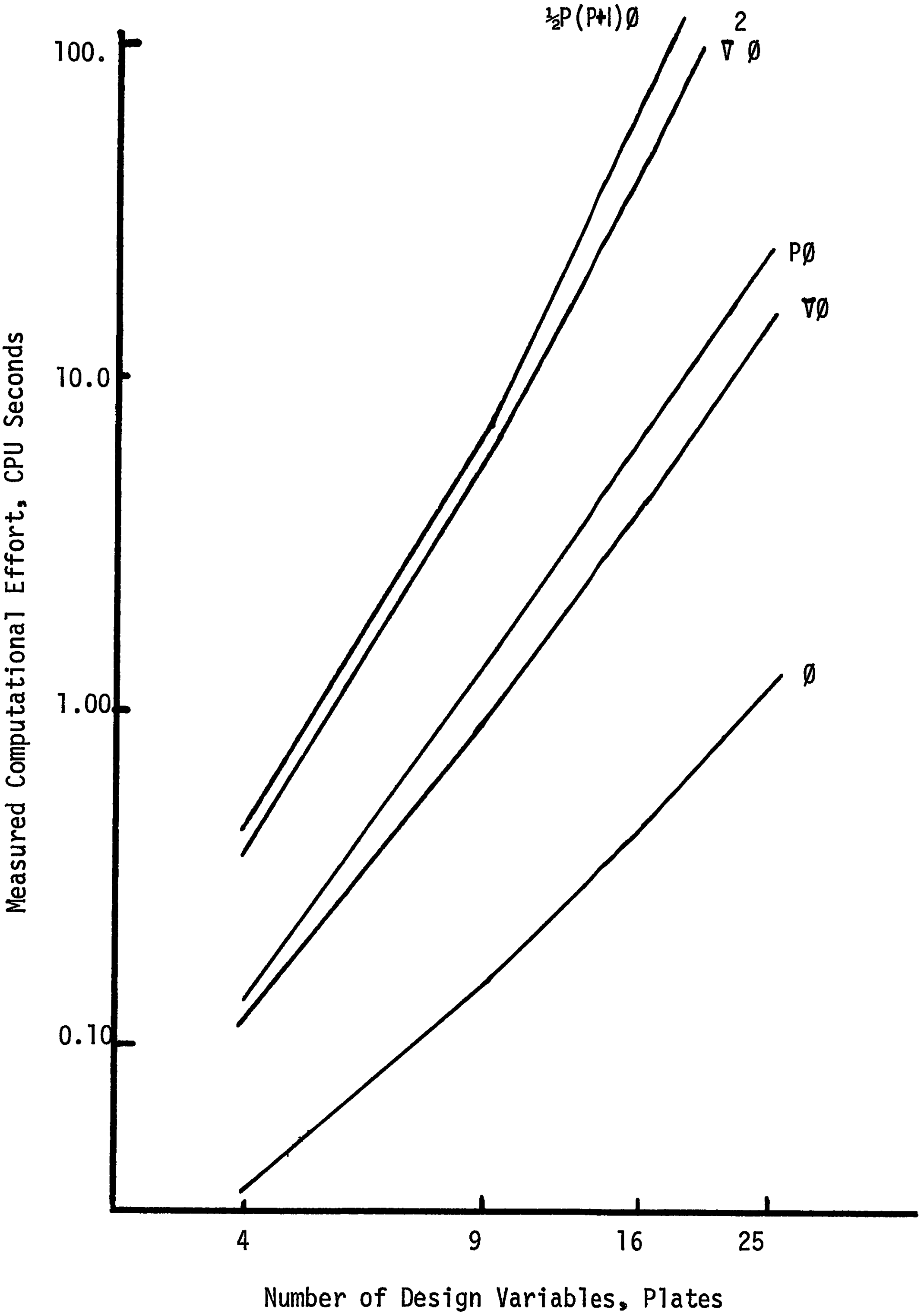

½P(P+1)Ø
2
∇ Ø
PØ
∇Ø
Ø
100.
10.0
1.00
0.10
4
9
16
25
Measured Computational Effort, CPU Seconds
Number of Design Variables, Plates

FIGURE 7.3.2

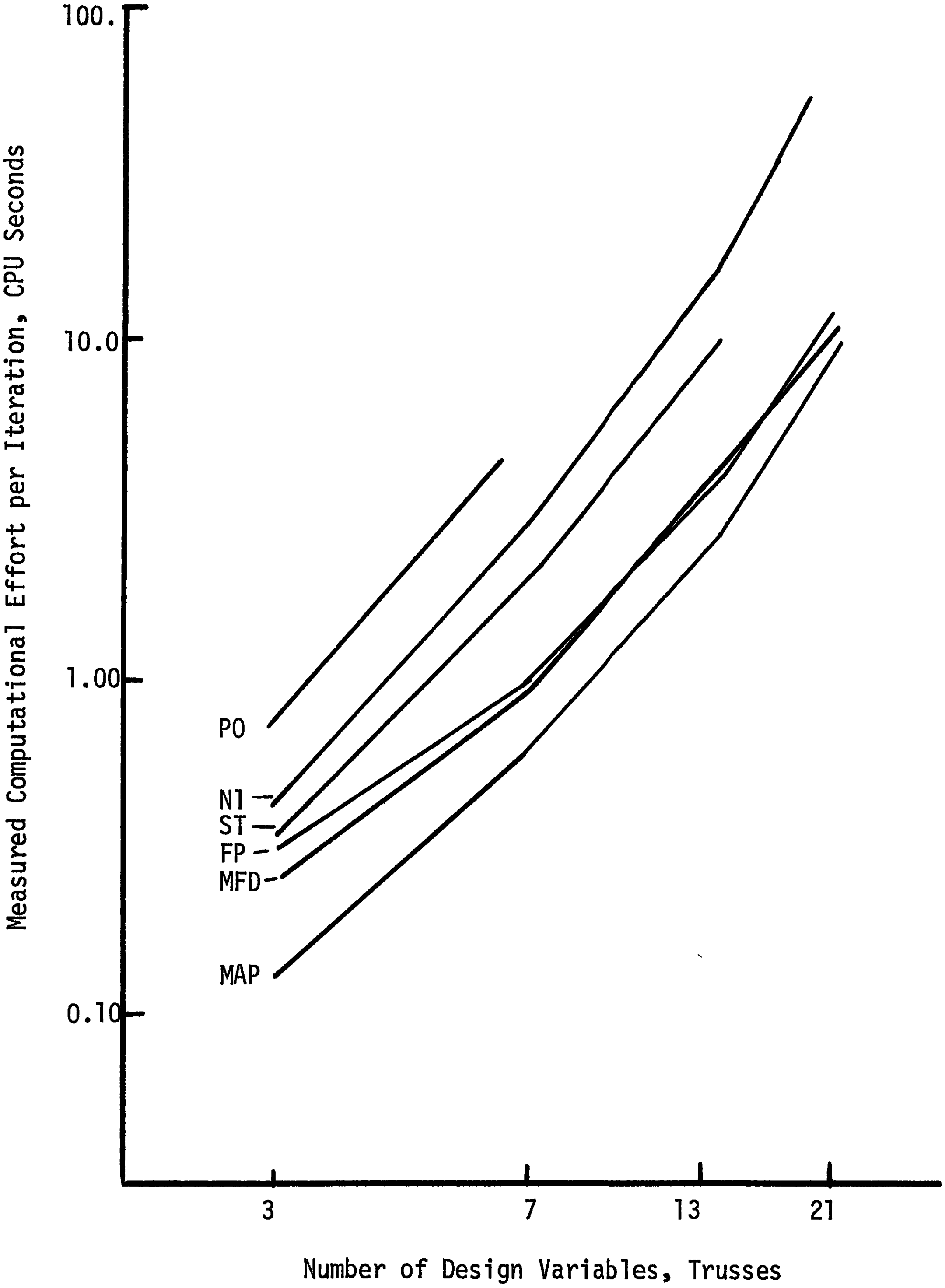
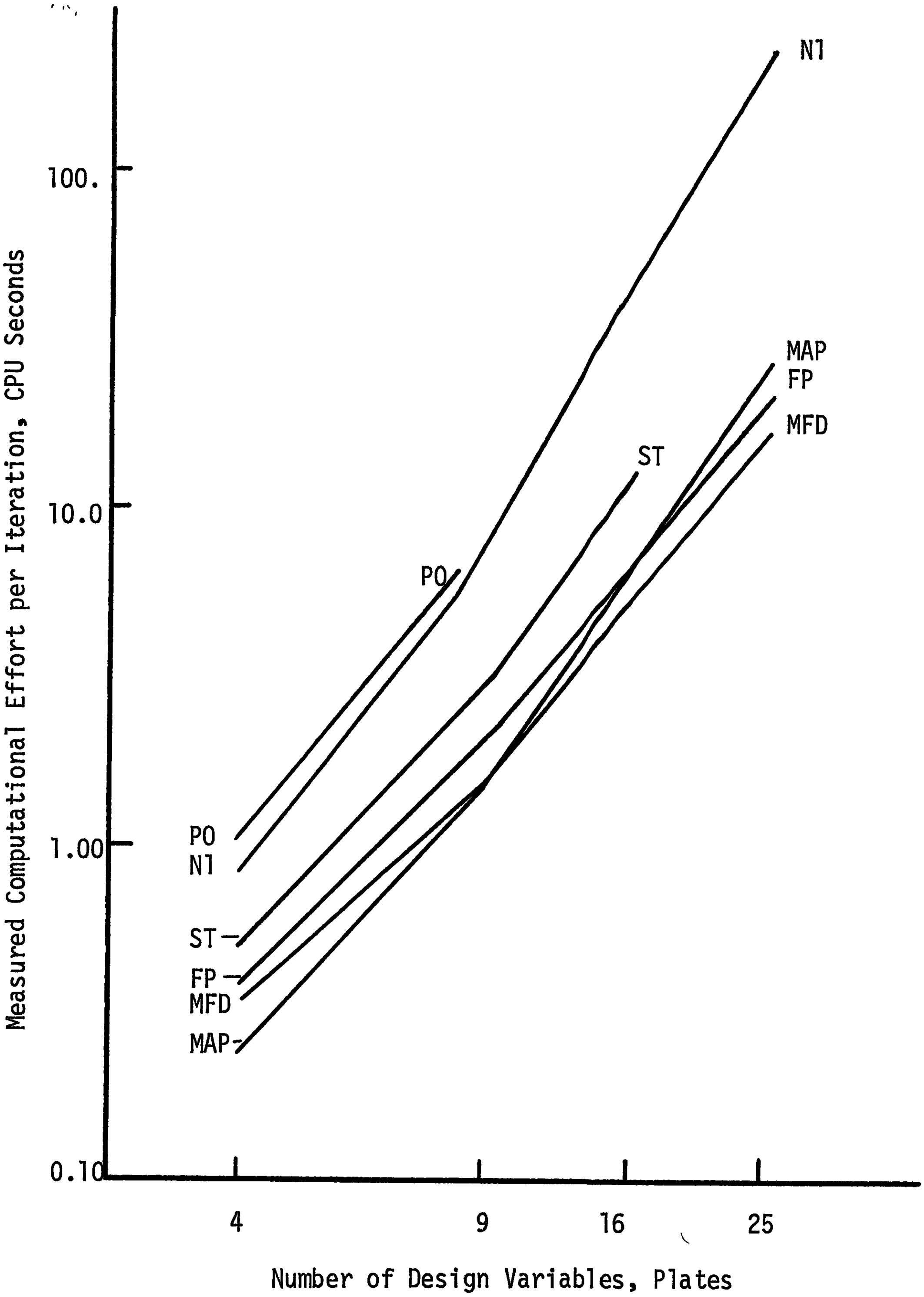

Measured Computational Effort per Iteration, CPU Seconds
100.
10.0
1.00
0.10
PO
N1
ST
FP
MFD
MAP
3
7
13
21
Number of Design Variables, Trusses

FIGURE 7.4.1



N1
100.
MAP
FP
MFD
ST
10.0
PO
PO
N1
1.00
ST
FP
MFD
MAP
0.10
4
9
16
25
Measured Computational Effort per Iteration, CPU Seconds
Number of Design Variables, Plates

FIGURE 7.4.2

are probably caused by the arbitrary assumption (made in chapter 5) of the number of iterations required by the LP algorithm to find the solution to the linearized problem.

7.5 Other results.

Table 7.5.1 shows the ratios of the CPU effort used in evaluating functions (F), derivatives (D) or in performing the optimisation operations (O) to the total CPU effort. The derivative-total effort ratio for Stewart's method was determined

Table 7.5.1: Measured effort ratios, $E_{X,Total}$, to achieve the NMW

ALG	X	P-BAR TRUSSES, P = 3	7	13	21	P-NODE PLATES, P = 4	9	16	25
MAP	F	0.32	0.23	0.16	0.14	0.22	0.13	0.08	0.05
	D	0.42	0.60	0.63	0.66	0.49	0.63	0.59	0.55
	O	0.26	0.17	0.21	0.20	0.29	0.24	0.33	0.40
MFD	F	0.36	0.38	0.47	-	0.27	0.17	0.10	-
	D	0.37	0.43	0.39	-	0.51	0.71	0.74	-
	O	0.27	0.18	0.14	-	0.22	0.12	0.16	-
N1	F	0.44	0.30	0.18	0.13	0.36	0.11	0.05	0.02
	D	0.52	0.68	0.81	0.86	0.59	0.88	0.94	0.97
	O	0.04	0.02	0.01	0.01	0.05	0.01	0.01	0.01
FP	F	0.70	0.61	0.55	0.53	0.54	0.47	0.34	0.27
	D	0.24	0.36	0.43	0.46	0.37	0.50	0.65	0.72
	O	0.06	0.03	0.02	0.01	0.09	0.03	0.01	0.01
ST	F	0.67	-	-	-	0.64	0.54	0.46	-
	D	0.26	-	-	-	0.28	0.45	0.53	-
	O	0.07	-	-	-	0.08	0.01	0.01	-
PO	F	0.94	-	-	-	0.94	0.98	-	-
	D	-	-	-	-	-	-	-	-
	O	0.06	-	-	-	0.06	0.02	-	-

by including in the derivative effort only those functions evaluations necessary for a forward FD derivative scheme. The effort used by the remaining function evaluations was used to determine the function-total effort ratio.

Table 7.5.2 shows the average number of function evaluations used per one-dimensional search for each of the algorithms. The high values reported for Stewart's method are a result of the assumption that only the forward FD scheme was used by the algorithm. As assumed in chapter 5, the average number of function evaluations used per one-dimensional search was approximately 2.5 when MFD was used and was approximately 6.5 when either N1, FP or PO was used.

Table 7.5.2: Average number of function evaluations per one-dimensional search.

ALG	P-BAR TRUSSES, P =				P-NODE PLATES, P =			
	3	7	13	21	4	9	16	25
MFD	1.8	2.9	5.3	3.0	1.8	1.4	1.2	-
N1	5.9	7.8	6.6	8.6	9.0	5.8	6.1	6.5
FP	6.7	6.0	5.7	6.0	5.5	6.0	5.2	5.2
ST	7.9	9.8	13.	-	8.4	11.	13.	-
PO	7.4	6.8	-	-	6.9	6.3	-	-

It can be seen in table 7.5.3, showing the algorithm-MAP effort ratios to reach the NMW, that for the test problems, SUMT in conjunction with N1, FP, ST or PO requires much more effort to reach the minimum than either MFD or MAP.

The effect of improvements to SUMT and the UOAs and of using finite difference derivatives with MFD and MAP are discussed in section 6 of this chapter.

Table 7.5.3: Measured algorithm-MAP effort ratios, $E_{ALG,MAP}$, to reach the NMW

ALG	P-BAR TRUSSES, P =				P-NODE PLATES, P =			
	3	7	13	21	4	9	16	25
MAP	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
MFD	3.62	3.98	4.20	-†	1.89	2.08	3.67	-†
N1	13.1	20.2	11.8	15.5	12.9	16.0	28.7	43.5
FP	14.2	19.9	13.0	15.4	8.28	10.1	11.9	14.6
ST	20.1	-	-†	-	13.0	23.2	26.8	-
PO	47.1	-†	-	-	36.8	70.4	-	-

Table 7.5.4 shows the ratio of the number of iterations made by an algorithm to reach the near minimum weight, to the number of design parameters. Table 7.5.5 shows the ratio of the number of iterations required by an algorithm to reach the NMW to the number of iterations required by MAP to reach the NMW.

7.6 Discussion.

From the results shown in table 7.5.3 which summarizes the relative performances of the algorithms on the structural problems, it would appear that MAP and MFD require less computational effort than SUMT. This section investigates the effects of:

Table 7.5.4: Ratios of the number of iterations required to reach the NMW to the number of design variables.

ALG	P-BAR TRUSSES, P =				P-NODE PLATES, P =			
	3	7	13	21	4	9	16	25
MAP	.667	.571	.764	.524	.500	.444	.250	.160
MFD	2.00	.157	2.00	-	1.00	1.00	1.00	-
N1	3.00	2.43	1.46	.952	1.75	1.33	.938	.560
FP	4.33	6.14	6.54	5.57	2.75	3.33	3.06	2.96
ST	5.00	-	-	-	3.00	5.00	3.56	-
PO	5.78	-	-	-	4.25	5.26	-	-

Table 7.5.5: Ratios of the number of iterations required by an algorithm to reach the NMW to the number required by MAP.

ALG	P-BAR TRUSSES, P =				P-NODE PLATES, P =			
	3	7	13	21	4	9	16	25
MAP	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
MFD	3.00	2.75	2.62	-	2.00	2.25	4.52	-
N1	4.50	4.26	1.91	1.82	3.50	3.00	3.75	3.50
FP	6.50	10.8	8.56	10.6	5.50	7.50	12.2	18.5
ST	7.50	-	-	-	6.00	11.3	14.2	-
PO	8.66	-	-	-	8.50	11.8	-	-

i. using a more efficient search technique with the UOAs of SUMT,

ii. using finite differences derivatives with MAP and MFD and

iii. having derivative-function effort ratios different from those of this study.

The number of function evaluations per one-dimensional search can be reduced by a search technique developed by Lund and recommended by Moe[22]. In the search, quadratic polynomial approximations of the original objective and all the constraint functions are fitted to three points, the initial and two other points, along the search direction. The polynomial approximations are combined to form a new transformed objective function for this search of SUMT. The minimum of the new objective function is found with little computational effort. The original transformed objective function is evaluated at the new point and the search is terminated. Therefore, only three function evaluations are required per search. The effect of using such a search technique is estimated in table 7.6.1 from the results in tables 7.5.1 to 7.5.3. In table 7.6.1

Table 7.6.1: Estimated algorithm - MAP effort ratios, $E_{ALG,MAP}$, to reach the NMW when Lund's search is used.

ALG	P-BAR TRUSSES, P =				P-NODE PLATES, P =			
	3	7	13	21	4	9	16	25
MAP	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
MFD	3.62	3.98	4.20	-	1.89	2.08	3.67	-
N1	10.2	16.8	10.7	14.7	9.88	15.2	27.8	43.1
FP	8.66	14.1	9.47	11.2	6.22	7.78	8.69	13.0
ST	11.7	-	-	-	5.69	14.0	17.2	-
PO	21.2	-	-	-	16.9	33.8	-	-

it can be seen that the effort required by the UOAs and SUMT relative to the effort required by MAP and MFD would be reduced by using Lund's technique. Therefore in the following work in this chapter, the values in table 7.6.1 will be used.

When finite differences are used to obtain derivatives, the effort necessary per iteration and the number of iterations required are greater than when derivatives are obtained by differentiation. The greater effort per iteration can be estimated from tables 7.3.3, 7.5.1 and 7.6.1. The greater number of iterations required can be estimated from table 7.5.5 by comparing the number of iterations used by Stewart's method with the number used by Fletcher-Powell's method. With the assumption that the number of iterations required is 25% greater than when derivatives are obtained by differentiating, table 7.6.2 gives estimates of the algorithm-MAP effort ratios to reach the near minimum weight when derivatives are obtained by finite differences.

Table 7.6.2: Estimated algorithm-MAP effort ratios, $E_{ALG,MAP}$ to reach the NMW when Lund's search and forward FD derivatives are used.

ALG	P-BAR TRUSSES, P =				P-NODE PLATES, P =			
	3	7	13	21	4	9	16	25
MAP	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
MFD	3.53	3.53	3.33	-	1.90	2.13	3.91	-
ST	7.67	-	-	-	4.14	8.80	10.0	-
PO	13.9	-	-	-	12.3	21.3	-	-

The derivative-function effort ratios for the function and derivative evaluation algorithms used are given in table 7.3.3. However, different function and derivative algorithms may give different derivative-function effort ratios which would affect the effort ratios to reach the near minimum weight but should not affect the path taken by the optimisation algorithm to get to the near minimum weight design. Table 7.6.3 estimates the effect on the algorithm-MAP effort ratios to reach the near minimum weight, of different derivative-function effort ratios. The values are determined from the following equations.

Let

$T_{\emptyset}$ = effort to evaluate a function,

$T_{\nabla\emptyset}$ = effort to evaluate a first derivative by differentiation,

$T_{\Delta\emptyset}$ = effort to evaluate a first derivative by forward finite differences, and

P = number of design variables.

Then

$$E_{\nabla\emptyset,\emptyset} = T_{\nabla\emptyset} \big/ T_{\emptyset} \; ; \; E_{\Delta\emptyset,\emptyset} = T_{\Delta\emptyset} \big/ T_{\emptyset} = P \qquad \ldots 7.6.4,$$

therefore

$$E_{\nabla\emptyset,\emptyset} = E_{\nabla\emptyset,\Delta\emptyset} \, P \qquad \ldots 7.6.5.$$

Similarly, let

$T_{\nabla^2\emptyset}$ = effort to evaluate a second derivative by differentiation,

$T_{\Delta^2\emptyset}$ = effort to evaluate a second derivative by forward finite differences.

Table 7.6.3: Estimated algorithm - MAP effort ratios, $E_{ALG,MAP}$, to reach the NMW for different values for the derivative-function effort ratio μ.

ALG	μ	P-BAR TRUSSES, P = 3	7	13	21	P-NODE PLATES, P = 4	9	16	25
MAP	all	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
MFD	1/P	4.83	4.43	4.06	-	3.22	3.62	7.28	-
	½	4.51	3.45	2.95	-	2.82	2.69	4.96	-
	1	3.88	3.07	2.73	-	2.44	2.43	4.62	-
	2	3.42	2.84	2.60	-	2.19	2.28	4.44	-
	10	2.96	2.64	2.49	-	1.95	2.15	4.29	-
N1	1/P	10.3	13.0	11.0	16.9	8.70	10.3	17.9	22.8
	½	10.4	14.9	10.7	15.7	8.95	12.5	25.2	35.4
	1	10.4	15.5	11.2	16.2	9.13	13.1	26.2	36.4
	2	10.4	15.9	11.4	16.5	9.25	13.4	26.8	37.0
	10	10.4	16.2	11.7	16.7	9.36	13.7	27.2	37.5
FP	1/P	10.0	16.7	13.2	16.4	8.50	11.6	18.9	28.6
	½	9.06	12.1	8.40	9.64	7.10	7.92	11.5	16.5
	1	7.55	10.5	7.58	8.95	5.96	6.95	10.6	15.4
	2	6.47	9.48	7.12	8.59	5.21	6.42	10.0	14.9
	10	5.36	8.60	6.73	8.29	4.47	5.94	9.57	14.4
ST	1/P	17.4	-	-	-	16.2	52.4	104.	-
	½	13.9	-	-	-	10.8	19.1	23.2	-
	1	8.71	-	-	-	6.51	10.5	12.3	-
	2	4.98	-	-	-	3.61	5.52	6.33	-
	10	1.13	-	-	-	0.79	1.15	1.30	-
PQ	1/P	30.1	-	-	-	39.4	123.	-	-
	½	24.1	-	-	-	26.3	44.9	-	-
	1	15.1	-	-	-	15.8	24.6	-	-
	2	8.62	-	-	-	8.78	13.0	-	-
	10	1.95	-	-	-	1.93	2.71	-	-

Then

$$E_{\nabla^2\phi,\phi} = T_{\nabla^2\phi} \Big/ T_{\phi} \; ; \; E_{\Delta^2\phi,\phi} = T_{\Delta^2\phi} \Big/ T_{\phi} = P\ (\ P+1\ )/2 \qquad \text{...7.6.6;}$$

therefore

$$E_{\nabla^2\phi,\phi} = E_{\nabla^2\phi,\Delta^2\phi}\ P\ (\ P+1\ )/2 \qquad \text{...7.6.7,}$$

Table 7.3.3 gives that

$$E_{\nabla\phi,\Delta\phi} \doteq E_{\nabla^2\phi,\Delta^2\phi} \qquad \text{...7.6.8.}$$

Therefore let

$$\mu \doteq E_{\nabla\phi,\Delta\phi} \doteq E_{\nabla^2\phi,\Delta^2\phi} \qquad \text{...7.6.9.}$$

Thus, if $\mu = 1/P$, then a derivative evaluation by differentiation requires as much effort as a function evaluation; if $\mu = 1$, then a derivative evaluation by differentiation requires as much effort as one obtained by forward finite differences; if $\mu = 2$, then a derivative evaluation by differentiation requires as much effort as one obtained by central finite differences. If $\mu = 4$, a higher order finite difference derivative may use less effort than and may be as accurate as a differentiation derivative. The computational effort required by the algorithms to perform one iteration is given by:

$$T_{MAP} = 1.33\ (\ T_{\phi} + T_{\nabla\phi}\ ) \qquad \text{...7.6.10,}$$

$$T_{MFD} = 1.25\ (\ 2.5\ T_{\phi} + T_{\nabla\phi}) \qquad \text{...7.6.11,}$$

$$T_{N1} = 1.02\ (\ 3.0\ T_{\phi} + T_{\nabla\phi} + T_{\nabla^2\phi}\ ) \qquad \text{...7.6.12,}$$

$$T_{FP} = 1.03 (3.0\, T_{\emptyset} + T_{\nabla\emptyset}) \qquad ...7.6.13,$$

$$T_{ST} = 1.03 (3.0\, T_{\emptyset} + T_{\Delta\emptyset}) \qquad ....7.6.14,$$

and

$$T_{PO} = 1.03 (3.0\, T_{\emptyset}) P \qquad ...7.6.15,$$

where the coefficients 1.33, 1.25, 1.02, 1.03, 1.03, 1.03 account for the effort used by the optimisation algorithms and were obtained from table 7.5.1.

Substituting the equations 7.6.4 to 7.6.9 into 7.6.10 to 7.6.15 gives:

$$T_{MAP} = 1.33 (1+\mu P) T_{\emptyset} \qquad ...7.6.16,$$

$$T_{MFD} = 1.25 (2.5 + \mu P) T_{\emptyset} \qquad ...7.6.17,$$

$$T_{N1} = 1.02 (3.0 + \mu P (P+3)/2) T_{\emptyset} \qquad ...7.6.18,$$

$$T_{FP} = 1.03 (3.0 + \mu P) T_{\emptyset} \qquad ...7.6.19,$$

$$T_{ST} = 1.03 (3.0 + P) T_{\emptyset} \qquad ...7.6.20,$$

and

$$T_{PO} = 1.03 (3.0) P\, T_{\emptyset} \qquad ...7.6.21.$$

Therefore, the effort per iteration ratios are

$$E_{MAP,MAP} = 1.0 \qquad ...7.6.22$$

$$E_{MFD,MAP} = \frac{1.25 (2.5 + \mu P)}{1.33 (1 + \mu P)} \qquad ...7.6.23$$

$$E_{N1,MAP} = \frac{1.02 (3.0 + \mu P (P + 3)/2)}{1.33 (1 + \mu P)} \qquad ...7.6.24$$

$$E_{FP,MAP} = \frac{1.03\ (\ 3.0 + \mu P\ )}{1.33\ (\ 1 + \mu P\ )} \qquad \ldots 7.6.25$$

$$E_{ST,MAP} = \frac{1.03\ (\ 3.0 + P\ )}{1.33\ (\ 1 + \mu P\ )} \qquad \ldots 7.6.26$$

$$E_{PO,MAP} = \frac{1.03\ (\ 3.0\ P\ )}{1.33\ (\ 1 + \mu P\ )} \qquad \ldots 7.6.27.$$

Equations 7.6.22 to 7.6.27 and the number of iterations ratios in table 7.5.5 were used to determine the estimated effort ratios in table 7.6.3. In table 7.6.3 it can be seen that when a differentiation derivative evaluation requires as much effort as a function evaluation, MAP would require less effort than any of the methods considered. MFD would reqqire more effort than MAP but less effort than the other methods considered.

When a differentiation derivative evaluation requires as much effort as a central difference derivative evaluation, MAP still requires the least effort.

When a differentiation derivative evaluation requires much more effort than a central difference derivative evaluation, the non-derivative methods require approximately as much effort as MAP and MFD. However it is unlikely that such differentiation derivatives would be used since high order polynomial approximations to the derivatives would require less effort and may be as accurate.

CHAPTER 8

CONCLUSION

8.1 Conclusions.

The test results of chapter 7 verify the estimates made in chapter 5 of the relative effort required by the function, derivative and optimisation algorithms used in this study. From these results the following conclusions can be drawn:

1. a first derivative evaluation requires much more effort than a function evaluation;
2. a second derivative evaluation requires much more effort than a first derivative evaluation;
3. finite difference derivatives require more computational effort than differentiation derivatives;
4. the effort to solve the LP problem for MAP or MFD is approximately equal to the effort to evaluate a first derivative;
5. the effort to generate a search direction for the UOAs, not including the necessary function and derivative evaluation effort, is approximately equal to the effort to perform a function evaluation.

Therefore, procedure 5.1.1 can give useful estimates of the CPU time involved in computations.

The preliminary results in chapter 6 show the effect of 'tuning' an algorithm by the adjustment of the arbitrary coefficients and parameters in the algorithms to reduce the computational effort expended. For the results reported in chapter 7, it is assumed that a comparable degree of 'tuning' has been achieved.

The results of chapter 7 show that all the methods selected can be used to solve the structural problem 1.2 or 1.3, although those methods which did not use differentiation derivatives were less effective than the other algorithms.

Table 8.1.1 shows the algorithms listed in increasing order of computational effort required. The table also shows the type of derivative evaluation to be used.

Table 8.1.1: The algorithms, listed in increasing order of computational effort required.

		TYPE OF DERIVATIVES TO BE USED			
NO	ALGORITHM	DIFFERENTIATION	FORWARD F.D.		CENTRAL F.D.
1	MAP	X	-		-
2	MAP	-	X		-
3	MAP	-	-		X
4	MFD	X	-		-
5	MFD	-	X		-
6	MFD	-	-		X
	SUMT + FP	X	-		-
8	SUMT + N1	X	-		-
9	SUMT + ST	-	X		-
10	SUMT + ST	-	-		X
	SUMT + N1	-	X	or	X
12	SUMT + PO	-	-		-

Table 8.1.1 summarizes the conclusions that can be drawn from the results of chapter 7.

8.2 Recommendations.

Table 8.1.1 lists the algorithms in increasing order of computational effort. However, other considerations, as given in chapter 1, may be more important than computational effort, in the selection of algorithms. Therefore, this section gives recommendations for the use of the algorithms on problems similar to 1.2 or 1.3.

If MFD is selected, the extra provision for increasing or decreasing the arbitrary coefficients to slow or speed the optimisation, may give computational savings.

If SUMT is selected and second derivatives are available, then a combined Newton and Fletcher-Powell method is suggested. The proposed method would proceed as in Newton's method for the first iteration, storing the inverse of the second derivative matrix, and then proceed as in Fletcher-Powell's method on succeeding iterations. However, if second derivatives are not available, Fletcher-Powell's method or a quasi-Newton method[30,31] is recommended. A search technique based on that of Lund, using directional derivatives when available, is preferred.

The ' Q ' transformation for SUMT is recommended as it obviates the difficulties associated with ρ . If, however, another transformation is chosen and requires ρ , then it is recommended that ρ_1 is found from equation 3.4.3 or from the following:

$$\rho_1 = c\,\rho_{3.4.9} \qquad \ldots 8.2.1$$

where $\rho_{3.4.9}$ is given by equation 3.4.9.

8.3 Further research.

A number of topics for further research arise from the results of this study:

1. establish the relative efficiency of MAP and MFD when FD derivatives are used instead of differentiation derivatives;
2. a. establish the relative efficiency of the Q transformation and other SUMT transformations;

   b. investigate the effect on the efficiency of alternative schemes for evaluating ρ_1 for SUMT;

   c. investigate the efficiency of the proposed Newton-Fletcher-Powell method used with SUMT;

   d. verify the efficiency of the search technique based on Lund's method used with SUMT;
3. investigate the efficiency of the Modified Interior Point methods; and
4. verify the conclusions for other types of structural problem.

8.4 Summary.

The subject of the thesis is a comparison of commonly-used algorithms applied to a class of structural optimisation problems. The types of structure under consideration are pin-jointed, plane trusses and plane stress plates. The optimisation problem is weight minimization of the structures subject to stress and design variable limits. Optimisation algorithms fall into one of three categories: Linearization, Feasible Direction and Transformation methods. Algorithms have been selected from each category in order to compare the computational effort required to solve the structural problems.

Comparison of the results of the computer runs shows that MAP requires the least effort, MFD requires more effort than MAP and SUMT requires most computational effort of the methods considered.

LIST OF REFERENCES

1. Wasiutynski Z. and Brandt A., 'The present state of knowledge in the field of Optimum design of Structures', Applied Mechanics Reviews, vol. 16, no. 5, May 1963, pp 341-350.

2. Sheu C.Y. and Prager W., 'Recent developments in Optimal Structural design', Applied Mechanics Reviews, vol. 21, no. 10, Oct. 1968, pp 985-992.

3. Schmit L.A., 'Structural Synthesis 1959-1969: A decade of progress', Japan - U.S. seminar on Matrix methods of Structural analysis and design, Tokyo, Aug. 1969.

4. Jacoby S.L.S., Kowalik J.S. and Pizzo J.T., Iterative Methods for Non-Linear Optimization problems, Prentice-Hall, Englewood Cliffs, N.J., 1972.

5. Zoutendijk G., 'Non-linear Programming: A numerical survey', J. SIAM Control, vol. 4, no. 1, 1966, pp 194-210.

6. Kelley J.E., 'The Cutting Plane Method for solving convex programs', J. SIAM, vol. 8, 1960, pp 703-712.

7. Cheney E.W. and Goldstein A.A., 'Newton's method for convex programming and Tchebycheff approximation', Numerical Mathematics, vol. 1, 1959, pp 253-268.

8. Griffith R.E. and Stewart R.A., 'A Non-linear Programming technique for the Optimization of continuous processing systems', Man. Sci., vol. 7, 1961, pp 379-392.

9a. Rosen J.B., 'The Gradient Projection method for Non-linear Programming, part I, Linear constraints', J. SIAM, vol. 8, 1960, p 181 ff.

9b. Rosen J.B., 'The Gradient Projection method for Non-linear Programming, part II, Non-linear constraints', J. SIAM, vol. 9, 1961, p 514 ff.

10. Gellatly R.A. and Gallagher R.H., 'A procedure for automated minimum weight Structural design', Aeronautical Quarterly, Aug. 1966, pp 216-230.

11. Zoutendijk G., Methods of Feasible Directions; A Study in Linear and Non-linear Programming, Elsevier Publishing Co., 1960.

12. Fiacco A.V. and McCormick G.P., Non-linear Programming: Sequential Unconstrained Minimization Techniques, John Wiley and Sons, Inc., 1968.

13. Himmelblau D.M., 'A uniform evaluation of Unconstrained Optimization techniques', presented at the Dundee Optimization Conference, June 1971, proceedings to be published, ed: Lootsma F.A.

14. Fletcher R., 'Function minimization without evaluating derivatives - a review', Computer Journal, vol. 8, no. 1, 1965, pp 33-41.

15. Fox R.L., Optimization methods for Engineering design, Addison - Wesley Publishing Co., 1971, pp 97-101.

16. Fletcher R. and Powell M.J.D., 'A rapidly convergent descent method for minimization', Computer Journal, vol. 6, 1963, pp 163-168.

17. Stewart G.W., 'A modification of Davidon's minimization method to accept difference approximations of derivatives', J. of the Association for Computing Machinery, vol. 14, no. 1, Jan. 1967, pp 72-83.

18. Powell M.J.D., 'An efficient method for finding the minimum of a function of several variables without calculating derivatives', Computer Journal, vol. 7, 1964, pp 155-162.

19. Kiefer, J., 'Optimum Sequential search and approximation methods under minimum regularity assumptions', J. SIAM, vol. 5, 1957, pp 105-136.

20. Davidon W.C., 'Variable metric method for minimization', A.E.C. Research and Development report, ANL - 55990 (rev.), 1959.

21. Russell D., Optimization Theory, W.A. Benjamin Inc., 1970, p 19.

22. Gallagher R.H. and Zienkiewicz O.C., (eds), Optimum Structural design, Theory and applications, John Wiley and Sons, 1973, pp 143-175, Moe J., 'Penalty Function methods'.

23. Fiacco A.V. and McCormick G.P., 'Computational Algorithm for the Sequential Unconstrained Minimization Technique for Non-linear Programming', Man. Sci., vol. 10, no. 4, July 1964, pp 601-617.

24. Hildebrand F.B., Introduction to Numerical Analysis, McGraw-Hill Book Co., Inc., 1956, pp 60-64.

25. Beale E.M.L., Mathematical Programming in Practice, Pitman, London, 1971.

26. Przemieniecki J.S., Theory of Matrix Structural Analysis, McGraw-Hill Book Co., Inc., 1968.

27. IBM System / 360 and System / 370 FORTRAN IV language, form GC 28 - 6515 - 8.

28. IBM System / 360 model 67 Functional characteristics, file no. S360 - 01, form A27 - 2719 - 0, pp 43-53.

29. Schmit, Jr. L.A., 'Structural design by systematic Synthesis', 2nd ASCE conference on Electronic Computation, Pittsburgh, Pa., Sept. 1960, pp 105-132.

30. Gill P.E. and Murray W., 'Quasi-Newton methods for unconstrained optimization', J. of the Inst. of Mathematics and its applications, vol. 9, 1972, pp 91-108.

31. Fletcher R., 'A new approach to Variable Metric algorithms', Computer Journal, vol. 13, no. 3, 1970, pp 317-322.

APPENDIX

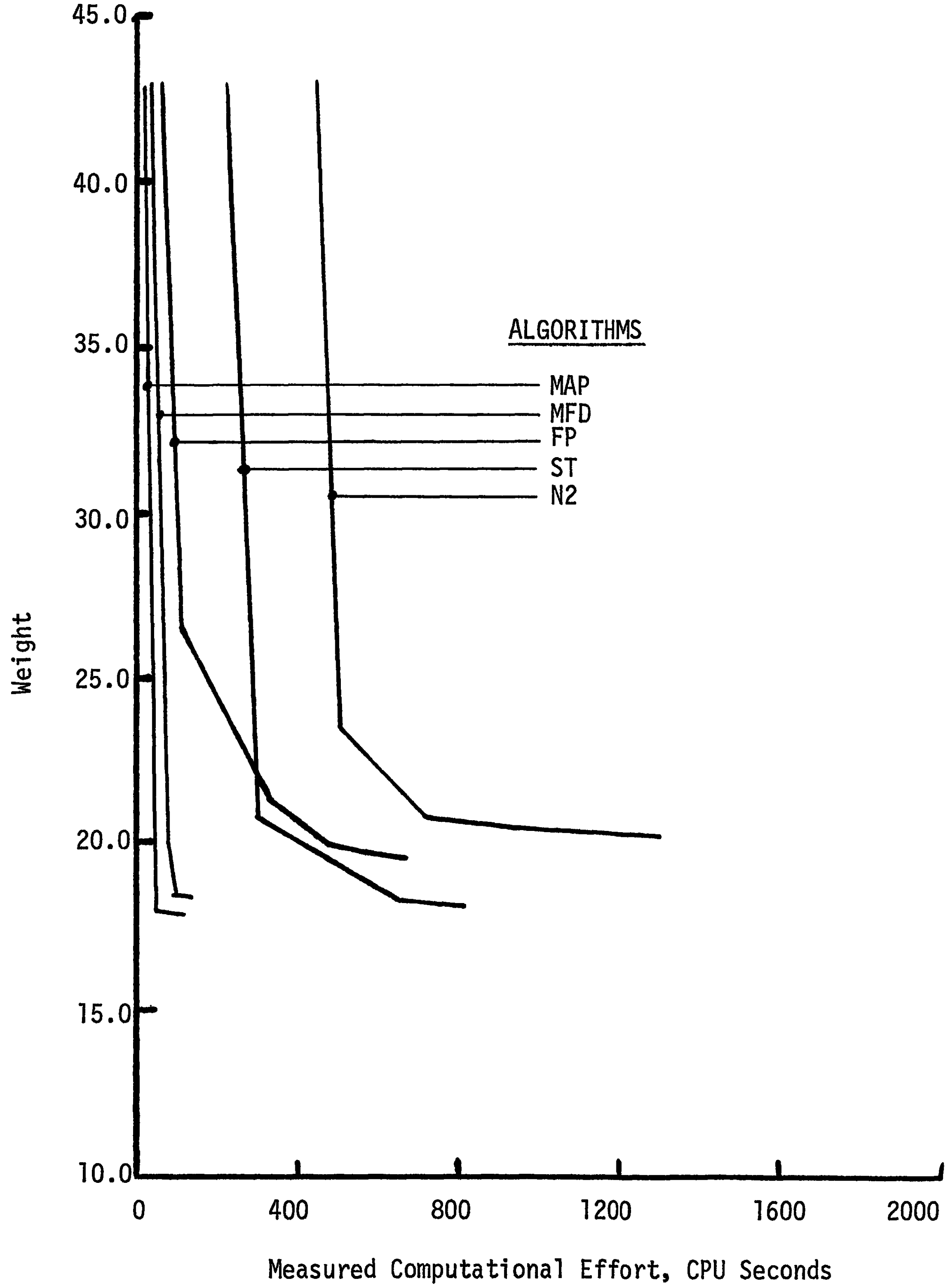

45.0
40.0
35.0
30.0
25.0
20.0
15.0
10.0
Weight
ALGORITHMS
MAP
MFD
FP
ST
N2
0
400
800
1200
1600
2000
Measured Computational Effort, CPU Seconds

FIGURE 10.1

```
C      STRUCTURAL OPTIMISATION PROGRAM
C
C      SYMBOLS USED
C
C
C      X()        = X NODE COORDINATE
C      Y()        = Y NODE COORDINATE
C      F()        = APPLIED LOADS MATRIX
C      P()        = MATRIX OF DISPLACEMENTS
C      EE         = MODULUS OF ELASTICITY
C      EENU       = POISSONS RATIO
C      RHO        = DENSITY
C      AK()       = MATRIX SAVING ELEMENT STIFFNESS MATRICES
C      XL()       = MATRIX WHICH MAPS NODAL THICKNESSES INTO WEIGHTS FOR
C                   PLATES OR MEMBER AREAS INTO WEIGHTS FOR RODS
C      STRS()     = MATRIX WHICH MAPS NODAL DISPLACEMENTS INTO STRESSES
C      S()        = MEMBER STRESSES
C      DSDT()     = FIRST DERIVATIVES OF STRESSES
C      ISITP      = 1 IF PLATE PROBLEM, 2 IF ROD PROBLEM
C      N          = NUMBER OF NODES
C      M          = NUMBER OF MEMBERS
C      NB         = NUMBER OF BOUNDARY CONDITIONS
C      NOD1()     = FIRST NODE NUMBER OF FINITE ELEMENT
C      NOD2()     = SECOND NODE NUMBER OF FINITE ELEMENT
C      NOD3()     = THIRD NODE NUMBER OF FINITE ELEMENT
C      IB()       = MATRIX OF DELETED FREEDOM INFORMATION
C      NK         = SIZE OF STIFFNESS MATRIX IF IN BLOCK
C      NLC        = NUMBER OF LOAD CASES
C      NT         = NUMBER OF TERMS IN EK()
C      IBW        = BAND WIDTH OF STIFFNESS MATRIX
C      NTIM()     = NUMBER OF TERMS IN EACH ROW STIFFNESS MATRIX
C      ISUM()     = LOCATION OF I,I STIFFNESS TERM IN EK()
C
C      ARSLTS()   = RESULTS MATRIX (REAL VALUES)
C      IRSLTS()   = RESULTS MATRIX (INTEGER VALUES)
C      IP         = DIRECTS LEVEL OF PRINTING
C      NONED      = NUMBER OF ONE DIMENSIONAL SEARCHES
C      NFE        = NUMBER OF FUNCTION EVALUATIONS
C      NGE        = NUMBER OF GRADIENT EVALUATIONS
C
C      VIRT       = VIRTUAL CPU TIME
C      TOTAL      = 'TOTAL' CPU TIME
C      OPTIM      = CPU TIME SPENT OPTIMISING
C      FUNTIM     = CPU TIME SPENT EVALUATING FUNCTIONS
C      DERTIM     = CPU TIME SPENT EVALUATING DERIVATIVES
C      TOTIM      = SUM OF OPTIM,FUNTIM AND DERTIM
C      T()        = MATRIX OF NODAL THICKNESS OR MATRIX OF MEMBER AREAS
C      TMAX       = MAXIMUM ALLOWABLE NODE THICKNESS FOR PLATE OR AREA FOR
C                   ROD
C      TMIN       = MINIMUM ALLOWABLE NODE THICKNESS FOR PLATE OR AREA FOR
C                   ROD
C      SIGA       = ALLOWABLE STRESS IN TENSION
C      SIGL       = ALLOWABLE STRESS IN COMPRESSION (A NEGATIVE NUMBER)
```

```
C      AL          = ALGORITHM TERMINATION PARAMETER : LB ON DESIGN CHANGE
C      FUNL        = ALGORITHM TERMINATION PARAMETER : LB ON FUNCTN CHANGE
C      TACTN       = RESOLUTION REQUIRED OF THE DESIGN VARIABLES
C      WTEST       = PROGRAM EXITED WHEN (WTI-WTIM1)/WTI.LT.WTEST
C      EST         = AN ESTIMATE OF THE MIN OF THE OBJECTIVE FUNCTION
C      EPS         = DISTINGUISHABILITY OF FUNCTION VALUES
C      EPM         = MACHINE RESOLUTION
C      TOL         = TOLERANCE ON TIGHTNESS OF CONSTRAINTS
C      FU          = UPPER BOUND ON CONSTRAINT VARIABLE
C      FL          = LOWER BOUND ON CONSTRAINT VARIABLE
C      FUN         = VALUE OF WEIGHT PLUS PENALTY FUNCTION = OBJECTIVE FUN
C      TREM()      = MATRIX WHICH HOLDS OLD DESIGN VARIABLES,WEIGHT,AND OF
C      DFDT()      = GRADIENT OF OBJECTIVE FUNCTION
C      H()         = WORK MATRIX USED BY UOA
C      WTIM1       = WEIGHT BEFORE A NEW ITERATION
C      WTI         = WEIGHT AFTER AN ITERATION
C      DUN         = AN ESTIMATE OF THE OBJECTIVE FUNCTION
C      RP          = WEIGHTING CONSTANT FOR PENALTY FUNCTION
C      PEN         = PENALTY FUNCTION ADDED TO WEIGHT TO GIVE OBJECTIVE FUN
C      FO,FN,FM    = SAVED FUNCTION VALUES OLD,NEW,MIDDLE
C      AO,AN,AM    = CORRESPONDING MOVES ALONG SEARCH DIRECTION
C      PO()        = DESIGN FOR FO
C      ALPHA       = MOVE LIMIT COEFFICIENT FOR MAP
C      NWORK       = NUMBER OF DESIGN VARIABLES (M FOR RODS, N FOR PLATES)
C      NRPV        = MAXIMUM NUMBER OF MAIN PROGRAM ITERATIONS ALLOWED
C      LIMIT       = MAXIMUM NUMBER OF ALGORITHM ITERATIONS ALLOWED
C      NOR         = CODE WHICH SPECIFIES THE OPTIMIZATION ALGORITHM USED
C                    NOR = 1, POWELL'S METHOD (POWL)
C                    NOR = 2, STEWART'S METHOD (STEW)
C                    NOR = 3, FLETCHER-POWELL'S METHOD (FLEP)
C                    NOR = 4, MODIFIED INTERIOR POINT METHOD (MIP)
C                    NOR = 5, METHOD OF APPROXIMATE PROGRAMMING (MAP)
C                    NOR = 6, METHOD OF FEASIBLE DIRECTIONS (MFD)
C                    NOR = 7, NEWTON'S METHOD   (NEWT)
C                    NOR = 8, QUADRATIC PROGRAMMING (QP)
C                    NOR = 9, NEW PROBLEM TO BE READ IN
C                    NOR = 10, END OF JOBS
C      ISRCH       = MAXIMUM NUMBER OF CURVE FITS PERMITTED IN ONED
C      IOPTS       = NUMBER OF ITERATIONS PERFORMED BY MAIN PROGRAM
C      IER         = 0 NO CONVERGENCE IN ALGORITHM
C                  = 1 CONVERGENCE
C                  = 2 MAX NO OF ITERATIONS
C      IHE         = 1 YIELDS FIRST DERIVATIVES ONLY
C                  = 2 YIELDS FIRST AND SECOND DERIVATIVES
C      IGH         = CODE FOR EFFLD
C      KOUNT       = NUMBER OF ITERATIONS PERFORMED BY ALGORITHM
C      NUSE        = NUMBER OF TIMES A SEARCH DIRECTION HAS BEEN USED
C      NSRCH       = CODE FOR SEARCH WITH POWELL'S METHOD
C      KODER()     = CODE FOR STEWART'S METHOD
C      ICOEF()     = VARIABLE ASSOCIATED WITH COLUMN IN A-MATRIX
C      IREM()      = ROW DESIGNATION OF ZERO B'S
```

```
C       NCOL        = NUMBER OF COLUMNS IN THE COEFFICIENT MATRIX FOR PRMDUL
C       NROW        = NUMBER OF   ROWS  IN THE COEFFICIENT MATRIX FOR PRMDUL
C
C       D2FDT2()    = HESSIAN OF OBJECTIVE FUNCTION
C       EK()        = STRUCTURAL STIFFNESS MATRIX
C       EKL()       = ELEMENT STIFFNESS MATRIX
C       Q()         = MATRIX WHICH SAVES NODAL DEFLECTIONS
C       R()         = MATRIX WHICH SAVES APPLIED LOADS
C       U()         = WORK MATRIX (MOVES DISPLACEMENTS)
C       DUDT()      = FIRST DERIVATIVES  OF DISPLACEMENTS
C       SS()        = MATRIX WHICH SAVES MEMBER STRESSES
C       SPACE()     = DUMMY ARRAY IN COMMON BLOCK 'WORK'
C
C       A()         = COEFFICIENT MATRIX FOR PRMDUL
C
C       PSI         = CONSTRAINT WEIGHTING CONSTANT FOR MFD
C       KM1()       = INDICES OF CONSTRAINTS HIT ON THE PREVIOUS MFD ITERN
C       KM2()       = INDICES OF CONSTRAINTS HIT ON ALL PREVIOUS MFD ITERNS
C
C       NOTE
C
C       1 .  LOAD DATA : SUBROUTINE DAT : IN = INDEX OF NODE , IC=1 FOR
C            FORCE IN X DIRECTION , IC=2 FOR FORCE IN Y DIRECTION , AMNT
C            AMOUNT OF LOAD ;
C       2 .  NOD1().LT.NOD2().LT.NOD3() ;
C       3 .  BOUNDARY CONDITION DATA :  X DIRECTION FREEDOMS DELETED :
C            ENTER NODE NUMBER , Y DIRECTION FREEDOMS DELETED : ENTER
C            1000 + NODE NUMBER ;
C       4 .  DIMENSION OF EK() = IBW*(NK-IBW/2+1/2) ;
C       5 .  SUBROUTINE  GELS IS AN IBM SSP SUBROUTINE ;
C       6 .  A()-SPACE() REPLACES D2FDT2()-SPACE() IN COMMON WORK FOR
C            SUBROUTINES MAP,MFD,PRMDUL AND SIMP ;
C
C       ******************************************************************

COMMENT : MAIN PROGRAM AND SUMT

        REAL*8 DATE,TIME
        INTEGER VIRT,TOTAL,OPTIM,FUNTIM,DERTIM,TOTIM
        COMMON/DATA/X(40),Y(40),F(80,5),P(80,5),EE,EENU,RHO,AK(1260),XL(60
       *),STRS(180,6),S(60,4,5),DSDT(60,5,60),ISITP,N,M,NB,NOD1(60),NOD2(6
       *0),NOD3(60),IB(80),NK,NLC,NT,IBW,NTIM(80),ISUM(80)
        COMMON/PRINT/ARSLTS(30,30),IRSLTS(30,30),IP,NONED,NFE,NGE
        COMMON/TIME/VIRT,TOTAL,OPTIM,FUNTIM,DERTIM,TOTIM
        COMMON/OPT/T(60),TMAX,TMIN,SIGA,SIGL,AL,FUNL,TACTN,WTEST,EST,EPS,
       *EPM,TOL,FU,FL,FUN,TREM(62),DFDT(60),H(2010),WTIM1,WTI,DUN,RP,PEN,
       *FO,FN,FM,AO,AN,AM,PO(60),ALPHA,NWORK,NRPV,LIMIT,NOR,ISRCH,IOPTS,IE
       *R,IHE,IGH,KOUNT,NUSE,NSRCH,KODER(60),ICOEF(380),IREM(40),NCOL,NROW
        COMMON/WORK/DEL(60),D2FDT2(60,60),D2FDA2(61),EK(3280),EKL(21),Q(80
       *,5),R(80,5),U(6,5),DUDT(80,60),SS(60,4,5),SPACE(510)
        COMMON/Z/PSI,KM1(280),KM2(280)
        DIMENSION TTT(62,12),TINIT(60),CL(12),CCL(12)
```

```
C *** THIS IS THE MAIN PROGRAM WHICH DIRECTS OPTIMIZATION OF A PLANE
C *** STRESS PROBLEM
    1 FORMAT(1H1)
    2 FORMAT(10H WEIGHT = ,E15.4)
    3 FORMAT (' INITIAL VALUE OF RP = ',E15.6)
    4 FORMAT(10H WEIGHT = ,E15.4,7H AFTER ,I3,14H OPTIMIZATIONS)
    5 FORMAT(' WEIGHT NOT CHANGING MUCH SO ALGORITHM TERMINATED')
    6 FORMAT(' MAXIMUM NUMBER OF UNCONSTRAINED OPTIMIZATIONS ALLOWED HAS
     X BEEN REACHED.  WE HAVE DONE',I3,' OPTIMIZATIONS')
    7 FORMAT(' WE ARE BEGINNING AN UNCONSTRAINED OPTIMIZATION PROGRAM WI
     CTH RP = ',E15.4)
    8 FORMAT(' WEIGHT INCREASING, ALGORITHM TERMINATED')
    9 FORMAT(' MATRIX T(I)'/6X,4HNODE,11X,4HT(I)/)
   10 FORMAT(I10,E15.4)
   11 FORMAT(' ERROR CODE FROM OPTIMIZATION ROUTINE = ',I3)
   12 FORMAT(/'OINITIAL VALUES FOR ALGORITHM CONTROL PARAMETERS'/'ORESOL
     *UTION    ',E15.6/'OREL CHANGE WT ',E15.6/'OREL CHANGE FUN',E15.6/'
     *OREL CHGE DSIGN',E15.6)
   13 FORMAT('OCPU TIMES ARE VIRTUAL CPU TIMES IN MICRO-SECONDS .')
   14 FORMAT(' OPTIM PERFORMED ',I10,' ONE DIMENSIONAL SEARCHES')
   15 FORMAT('OINITIAL RP COEFFICIENT = ',E15.6)
   16 FORMAT(' FUN = ',E15.4)
   17 FORMAT(' OPTIM PERFORMED ',I10,' FUNCTIONAL EVALUATIONS')
   18 FORMAT(' OPTIM PERFORMED ',I10,' GRADIENT EVALUATIONS')
   19 FORMAT(' BEGINNING ITERATION ',I5,' WE HAVE WEIGHT = ',E15.4,
     X' FUN = ',E15.4)
   20 FORMAT(' AFTER ITERATION ',I5,' WITH RP = ',E15.4,' WE HAVE '/
     X' WEIGHT = ',E15.4/' FUN = ',E15.4)
   21 FORMAT(I3)
   22 FORMAT(' UNRESTRAINED OPTIMIZATION ALGORITHM NOT SPECIFIED' )
   23 FORMAT('ORP REDUCTION RATE COEFFICIENT = ',E15.6)
   24 FORMAT(1X,I3,5E15.7)
   25 FORMAT(' DESIGN NOT CHANGING MUCH - ALGORITHM TERMINATED ')
   26 FORMAT(1X,3I10)
   27 FORMAT(1X,E15.8)
   28 FORMAT('ORESULTS FOR',I3,' PARAMETER PROBLEM USING ALGORITHM NO',
     *I3,'. DATE OF RUN ',A8,' TIME ',A8)
   29 FORMAT('OEND ITERATION ',7I15)
   30 FORMAT('OTOTAL NUMBER OF '//' ONE DIM SRCHS ',7I15)
   31 FORMAT('OFUNCTION EVALS',7I15)
   32 FORMAT('ODERIVATIVES    ',7I15)
   33 FORMAT('1 ALGORITHM CODE = NOR =',I3)
   34 FORMAT('OVALUE OF'/'OFUNCTION       ',7E15.6)
   35 FORMAT('OWEIGHT         ',7E15.6)
   36 FORMAT('OCPU TIMES FOR'/ 'OFUNCTION EVALS',7I15)
   37 FORMAT('ODERIVATIVES    ',7I15)
   38 FORMAT('OOPTIMIZING     ',7I15)
   39 FORMAT('OSUM OF TIMES   ',7I15)
   40 FORMAT('OMAXIMUM NO OF'/ 'OITERATIONS/RP ',7I15)
   41 FORMAT('OQUAD FITS/SRCH',7I15)
   42 FORMAT('OFEASIBILITY    ',7I15)
   43 FORMAT('OWEIGHT(SCALED)',7E15.6)
```

```
   44 FORMAT('OESTIMATED FUN.',7E15.6)
   45 FORMAT('OALPHA MOVE LIMIT COEFFICIENT = ',E15.6)
   46 FORMAT('OPSI CONSTRAINT WEIGHTING COEFFICIENT = ',E15.6)
C
  100 CONTINUE
      WRITE(6,1)
      CALL INIT
      NWORK=N
      IF(ISITP.EQ.2)NWORK=M
      NP=NWORK
      NP1=1+NP
      NP2=2+NP
      NRP1=1+NRPV
C *** SAVE DATA ***
      DO 125 I=1,NWORK
  125 TINIT(I)=T(I)
      SAL=AL
      SFL=FUNL
      STN=TACTN
      EPS=1.E-05
      EPM=1.E-06
      CCC=1./160.
      PSI=.1
C
C *** SET UP OPTIMIZATION ***
  200 CONTINUE
      CALL TIMER(DATE,TIME,VIRT,TOTAL)
      TOL=10.*EPM*SIGA
      IF(NOR.NE.6)GOTO 210
      TOL=.01*SIGA
      KU=2*(M*NLC+NP)
      DO 205 K=1,KU
      KM1(K)=0
      KM2(K)=0
  205 CONTINUE
  210 CONTINUE
      OPTIM=0
      FUNTIM=0
      DERTIM=0
      TOTIM=0
      NFE=0
      NGE=0
      NONED=0
      IOPTS=0
      AN=0.
      FO=0.
      NFE=1
      CALL SOLVE
C *** CALCULATE WEIGHT ***
      WRITE(6,1)
      WTIM1=0.
      DO 225 I=1,NWORK
```

```
  225 WTIM1=WTIM1+XL(I)*T(I)
      IF(IP.LT.0)GOTO 1225
      WRITE(6,2)WTIM1
 1225 CONTINUE
      IF(IOPTS.NE.0)GOTO300
      FUN=WTIM1
      DUN=FUN
      RP=0.
      CALL TIMER(DATE,TIME,VIRT,TOTAL)
      FUNTIM=FUNTIM+VIRT
      DO 245 I=1,NWORK
  245 TREM(I)=T(I)
      TREM(NP2)=WTIM1
      IF(NOR.EQ.5.OR.NOR.EQ.6)GOTO 300
C *** CALCULATE PENALTY
      DUM1=0.
      DUM2=0.
      DO250I=1,NWORK
  250 DUM1=DUM1+1./(TMAX-T(I))+1./(T(I)-TMIN)
      DUM1=DUM1*(TMAX-TMIN)
      DO 275 I=1,M
      DO 275 LC=1,NLC
  275 DUM2=DUM2+1./(SIGA-S(I,4,LC))+1./(S(I,4,LC)-SIGL)
      PEN=(SIGA-SIGL)*DUM2+DUM1
C *** CALCULATE INITIAL RP ***
C *** FIACCO AND MC CORMICK ***
      RR=.025*WTIM1/PEN
      RP=1.
      FUN=PEN
      IHE=1
      DUM1=0.
      DUM2=0.
      DUM3=0.
      DO 276 I=1,NWORK
      SPACE(I)=XL(I)
      DUM1=DUM1+XL(I)*XL(I)
  276 XL(I)=0.
      WRITE(6,27)DUM1
      IF(NOR.LE.2)GOTO 278
      CALL DERFUN
      GOTO 282
  278 DO 280 I=1,NWORK
  280 DEL(I)=.0001
      CALL DIFFUN
  282 CONTINUE
      DO 284 I=1,NWORK
      XL(I)=SPACE(I)
      DUM3=DUM3+DFDT(I)*DFDT(I)
  284 DUM2=DUM2+XL(I)*DFDT(I)
      WRITE(6,27)DUM2
      RP=-DUM2/DUM3
      WRITE(6,27)RP
```

```
      DUM5=DUM2*DUM2-DUM1*DUM3
      IF(DUM5.LE.0.)GOTO 289
      DUM5=SQRT(DUM5)/DUM3
      IF(RP)288,288,290
  288 RP=RP+DUM5
  289 IF(RP.LT.0.)GOTO 294
      GOTO 296
  290 IF(DUM5.LT.RP)RP=RP-DUM5
      GOTO 296
  294 RP=RR
      WRITE(6,27)RP
  296 CC=RP*PEN/WTIM1
      FUN=WTIM1+PEN*RP
      WRITE(6,16)FUN
      DUN=FUN
      TREM(NP1)=FUN
      C1=CC
      EST=.9*WTIM1
      DO 299 I=1,NRPV
      CCL(I)=1.
  299 CL(I)=1.
C
C *** CALL OPTIMIZATION ROUTINE ***
  300 CONTINUE
      IOPTS=1+IOPTS
      IF(IP.LT.0)GO TO 1325
      WRITE(6,1)
      WRITE(6,7)RP
      WRITE(6,19)IOPTS,WTIM1,FUN
 1325 CONTINUE
      GOTO(401,402,403,404,405,406,407,408),NOR
      WRITE(6,22)
      CALL EXIT
  401 ISRCH=6+IOPTS/3
      LIMIT=NP*(2+IOPTS/2)
      CALL POWL
      GOTO 425
  402 ISRCH=6+IOPTS/3
      LIMIT=NP*(2+IOPTS/2)
      CALL STEW
      GOTO 425
  403 ISRCH=6+IOPTS/3
      LIMIT=NP*(2+IOPTS/2)
      CALL FLEP
      GOTO 425
  404 CALL MIP
      GOTO 425
  405 ISRCH=10*NP
      LIMIT=1+IOPTS/3
      ALPHA=.2
      CALL MAP
      GOTO 425
```

```
  406 ISRCH=6+IOPTS/3
      LIMIT=4+IOPTS/2
      CALL MFD
      GOTO 425
  407 ISRCH=6+IOPTS/3
      LIMIT=(1+NP*(3+IOPTS))/2
      CALL NEWT
      GOTO 425
  408 CALL QP
      GOTO 425
  425 CONTINUE
C *** CALCULATE NEW WEIGHT ***
      WTI=0.
      DO450I=1,NWORK
  450 WTI=WTI+XL(I)*T(I)
      IF(IP.LT.0)GOTO1465
      WRITE(6,1)
      WRITE(6,20)IOPTS,RP,WTI,FUN
      WRITE(6,14)NONED
      WRITE(6,17)NFE
      WRITE(6,18)NGE
      WRITE(6,11)IER
      WRITE(6,9)
      WRITE(6,10)(I,T(I),I=1,NWORK)
      WRITE(6,1)
 1465 CONTINUE
      CALL TIMER(DATE,TIME,VIRT,TOTAL)
      OPTIM=OPTIM+VIRT
      TOTIM=OPTIM+FUNTIM+DERTIM
      I=IOPTS
C *** UPDATE RESULTS MATRICES ***
      IRSLTS(I,1)=NONED
      IRSLTS(I,2)=NFE
      IRSLTS(I,3)=NGE
      IRSLTS(I,4)=FUNTIM
      IRSLTS(I,5)=DERTIM
      IRSLTS(I,6)=OPTIM
      IRSLTS(I,7)=TOTIM
      IRSLTS(I,8)=LIMIT
      IRSLTS(I,9)=ISRCH
      IFEAS=0
      IF(NOR.EQ.5)CALL FEASQ(IFEAS)
      IRSLTS(I,10)=IFEAS
      ARSLTS(I,1)=FUN
      ARSLTS(I,2)=WTI
      SCALE=1.
      IF(IFEAS.EQ.0)GOTO 480
      DUM1=1./SIGA
      DUM2=1./SIGL
      DO 475 L=1,NLC
      DO 475 K=1,M
      DUM=DUM1
```

```
      IF(S(K,4,L).LT.0.)DUM=DUM2
      DUM=DUM*S(K,4,L)
      IF(DUM.GT.SCALE)SCALE=DUM
  475 CONTINUE
  480 CONTINUE
      ARSLTS(I,3)=SCALE*WTI
      ARSLTS(I,4)=EST
      WRITE(6,1)
      WRITE(6,28)NP,NOR,DATE,TIME
      WRITE(6,29)(I,I=1,IOPTS)
      WRITE(6,30)(IRSLTS(I,1),I=1,IOPTS)
      WRITE(6,31)(IRSLTS(I,2),I=1,IOPTS)
      WRITE(6,32)(IRSLTS(I,3),I=1,IOPTS)
      IF(NOR.EQ.5.OR.NOR.EQ.6)GOTO 1900
      WRITE(6,34)(ARSLTS(I,1),I=1,IOPTS)
      WRITE(6,44)(ARSLTS(I,4),I=1,IOPTS)
 1900 CONTINUE
      WRITE(6,35)(ARSLTS(I,2),I=1,IOPTS)
      IF(NOR.NE.5)GOTO 1902
      WRITE(6,42)(IRSLTS(I,10),I=1,IOPTS)
      WRITE(6,43)(ARSLTS(I,3),I=1,IOPTS)
 1902 CONTINUE
      WRITE(6,36)(IRSLTS(I,4),I=1,IOPTS)
      WRITE(6,37)(IRSLTS(I,5),I=1,IOPTS)
      WRITE(6,38)(IRSLTS(I,6),I=1,IOPTS)
      WRITE(6,39)(IRSLTS(I,7),I=1,IOPTS)
      WRITE(6,13)
      IF(NOR.EQ.5)GOTO 1905
      WRITE(6,40)(IRSLTS(I,8),I=1,IOPTS)
      WRITE(6,41)(IRSLTS(I,9),I=1,IOPTS)
 1905 CONTINUE
      WRITE(6,12)STN,WTEST,SFL,SAL
      IF(NOR.EQ.5.OR.NOR.EQ.6)GOTO 1910
      WRITE(6,15)CC
      WRITE(6,23)CCC
      GOTO 1920
 1910 CONTINUE
      IF(NOR.EQ.5)WRITE(6,45)ALPHA
      IF(NOR.EQ.6)WRITE(6,46)PSI
 1920 CONTINUE
C *** TEST FOR EXIT FROM  JOB ***
  500 CONTINUE
      IF(IOPTS.EQ.1)GOTO 600
      IF(NOR.EQ.5)GOTO 525
      IF(WTI.GT.WTIM1)GOTO 800
      IF(NOR.EQ.6)GOTO 525
      IF(FUN-WTI.LE.EPM*WTI)GOTO 820
  525 CONTINUE
      TEST=ABS((WTIM1-WTI)/WTI)
  550 IF(TEST.LT.WTEST)GOTO 820
      IF(NOR.NE.5)GOTO 575
      DO 560 I=1,NWORK
```

```
      TEST=ABS(TREM(I)-T(I))
      IF(TEST.GT.(AL*T(I)))GOTO 575
  560 CONTINUE
      GOTO 840
  575 IF(IOPTS.GT.NRPV)GOTO860
  600 CONTINUE
      WTIM1=WTI
      DO 610 I=1,NWORK
      TREM(I)=T(I)
  610 TTT(I,IOPTS)=T(I)
      TREM(NP1)=FUN
      TREM(NP2)=WTIM1
      TTT(NP1,IOPTS)=FUN
      TTT(NP2,IOPTS)=WTIM1
      RP=CCC*RP
      DUN=WTI+CCC*(FUN-WTI)
      FUN=DUN
      EST=WTIM1
      IF(NOR.GT.3.AND.NOR.NE.7)GO TO 300
      IF(IOPTS.EQ.1)GOTO 300
C *** EXTRAPOLATION ***
      IOP1=1+IOPTS
      IU=IOPTS-1
      CN=CCC**IOPTS
      CL(IOPTS)=(CN-1.)/(CCC-1.)
      DO 660 I=1,IU
      CL(I)=CL(I)*(CN*(1.-CN))/(CCC*CN-CCC**I)
  660 CONTINUE
      DO 670 J=1,NP2
  670 H(J)=0.
      DO 690 I=1,IOPTS
      DO 680 J=1,NP2
      H(J)=H(J)+CL(I)*TTT(J,I)
  680 CONTINUE
  690 CONTINUE
      DO 720 J=1,NWORK
  720 H(J)=H(J)-T(J)
      EST=H(NP2)
      NUSE=1
      NSRCH=1
      KOUNT=1
      NNOR=NOR
      NOR=1
      CALL ONED
      NOR=NNOR
      GOTO 300
C
C *** OPTIMIZATION COMPLETE ***
  800 CONTINUE
      WRITE(6,8)
      DO 805 I=1,NWORK
  805 T(I)=TREM(I)
```

```
      FUN=TREM(NP1)
      WTI=TREM(NP2)
      GOTO 900
  820 CONTINUE
      WRITE(6,5)
      GOTO 900
  840 CONTINUE
      WRITE(6,25)
      GOTO 900
  860 CONTINUE
      WRITE(6,6)IOPTS
      GOTO 900
  900 CONTINUE
C *** READ IN NEXT JOB ***
      READ(5,21)NOR
      WRITE(6,33)NOR
      IF(NOR.EQ.10)GOTO 999
      IF(NOR.EQ.9)GOTO100
      DO 950 I=1,NWORK
  950 T(I)=TINIT(I)
      AL=SAL
      FUNL=SFL
      TACTN=STN
      GOTO 200
  999 CALL EXIT
      END
```

COMMENT : OPTIMIZATION ALGORITHMS

```
      SUBROUTINE POWL
C *** NOR=1 ***
COMMON CARDS:PRINT,OPT,WORK
      IER=0
      KOUNT=0
      NZERO=0
      DO 90I=1,NWORK
      DO 85J=1,NWORK
   85 D2FDT2(I,J)=0.
   90 D2FDT2(I,I)=-1.
  100 CONTINUE
      KOUNT=KOUNT+1
      NUSE=1
      DO102J=1,NWORK
  102 PO(J)=T(J)
      ITST=0
      FO=FUN
      DELT=0.
C *** SEARCH IN THE N DIRECTIONS DEFINED BY D2FDT2
      DO108I=1,NWORK
      DO104J=1,NWORK
  104 H(J)=D2FDT2(J,I)
      FSAV=FUN
      NSRCH=I
      CALL ONED
      IF(AN.NE.O.)NZERO=0
      IF(AN.EQ.O.)NZERO=1+NZERO
      IF(NZERO.GE.NWORK)GOTO 136
      FTST=FSAV-FUN
      IF(DELT.GE.FTST)GOTO106
      DELT=FTST
C *** DELT IS LARGEST CHANGE IN OF DURING THE NWORK SEARCHES
C *** ITST IS ITERATION OF THE LARGEST CHANGE
      ITST=I
  106 CONTINUE
  108 CONTINUE
      DO110J=1,NWORK
  110 PN(J)=T(J)
      FN=FUN
C *** TEST TO SEE IF WE SEARCH IN SAME DIRECTION AGAIN
      DO112J=1,NWORK
  112 T(J)=(PO(J)+PN(J))/2.
      CALL FUNCT
      FM=FUN
      DUM=FO-2.*FM+FN
      IF(DUM.LT.O.)GOTO 120
      IF(FO-4.*FM+3.*FN.GT.O.)GOTO 116
      IF(2.*DELT*DUM.GE.(FO-FN)**2)GOTO 120
C *** POWL HAS DECIDED TO KEEP OLD DIRECTIONS AS TEST1.LT.TEST2
  116 CONTINUE
```

```
      DO118J=1,NWORK
      T(J)=PN(J)
  118 CONTINUE
      FUN=FN
      GOTO130
C *** WE WILL SEARCH IN PN - PO DIRECTION
  120 CONTINUE
      NUSE=0
      DO122J=1,NWORK
  122 H(J)=PN(J)-PO(J)
      DO124J=1,NWORK
  124 T(J)=PN(J)
      FUN=FN
      AO=-1.0
      NSRCH=NWORK+1
      CALL ONED
      IF(AN.NE.O.)NZERO=0
      IF(AN.EQ.O.)NZERO=1+NZERO
      IF(NZERO.GE.NWORK)GOTO 136
C *** GET NEW DIRECTION OF SEARCH
      NWM1=NWORK-1
      DO126I=ITST,NWM1
      D2FDA2(I)=D2FDA2(I+1)
      DO126J=1,NWORK
  126 D2FDT2(J,I)=D2FDT2(J,I+1)
      D2FDA2(NWORK)=D2FDA2(NWORK+1)
      DO128J=1,NWORK
  128 D2FDT2(J,NWORK)=H(J)
C *** DO WE TERMINATE
  130 CALL EXTEST
      IF(KOUNT.LT.NWORK)IER=0
      IF(IER.GT.0)GOTO 136
  132 DO134J=1,NWORK
  134 PO(J)=T(J)
      GOTO100
  136 CONTINUE
C *** NEITHER FUNCTION NOR VARIABLES    CHANGING MUCH SO WE TERMINATE
      RETURN
      END


      SUBROUTINE STEW
C *** NOR=2 ***
      CALL FLEP
      RETURN
      END


      SUBROUTINE FLEP
C *** NOR=3 ***
COMMON CARDS:PRINT,OPT,WORK
```

```
C           H       - WORKING STORAGE OF DIMENSION N*(N+7)/2.
      N=NWORK
      IER=0
      KOUNT=0
      N2=N+N
      N3=N2+N
      N31=N3+1
C        COMPUTE FUNCTION VALUE AND GRADIENT VECTOR FOR INITIAL ARGUMENT
      IHE=1
  100 IF(NOR.EQ.3)GOTO102
      DO 101 I=1,N
      DEL(I)=0.001
  101 KODER(I)=0
      CALL DIFFUN
      GOTO103
  102 CALL DERFUN
  103 CONTINUE
C        RESET ITERATION COUNTER AND GENERATE IDENTITY MATRIX
    1 K=N31
      DO 4 J=1,N
      H(K)=1.
      HINV(J)=1.
      NJ=N-J
      IF(NJ)5,5,2
    2 DO 3 L=1,NJ
      KL=K+L
    3 H(KL)=0.
    4 K=KL+1
C        START ITERATION LOOP
    5 KOUNT=KOUNT +1
C *** SAVE F ARG VECTOR GRAD VECTOR ***
      FO=FUN
      DO 9 J=1,N
      K=N+J
      H(K)=DFDT(J)
      K=K+N
      H(K)=T(J)
      PO(J)=T(J)
C        DETERMINE DIRECTION VECTOR H
      K=J+N3
      TT=0.
      DO 8 L=1,N
      TT=TT-DFDT(L)*H(K)
      IF(L-J)6,7,7
    6 K=K+N-L
      GO TO 8
    7 K=K+1
    8 CONTINUE
    9 H(J)=TT
C        CHECK WHETHER FUNCTION WILL DECREASE STEPPING ALONG H.
      DY=0.
      HNRM=0.
```

```
      GNRM=0.
C        CALCULATE DIRECTIONAL DERIVATIVE AND TESTVALUES FOR DIRECTION
C        VECTOR H AND GRADIENT VECTOR DFDT
      DO 10 J=1,N
      HNRM=HNRM+ABS(H(J))
      GNRM=GNRM+ABS(DFDT(J))
   10 DY=DY+H(J)*DFDT(J)
C        REPEAT SEARCH IN DIRECTION OF STEEPEST DESCENT IF DIRECTIONAL
C        DERIVATIVE APPEARS TO BE POSITIVE OR ZERO.
      IF(DY)11,50,50
C        REPEAT SEARCH IN DIRECTION OF STEEPEST DESCENT IF DIRECTION
C        VECTOR H IS SMALL COMPARED TO GRADIENT VECTOR DFDT
   11 IF(HNRM/GNRM-EPS)50,50,12
C        SEARCH MINIMUM ALONG DIRECTION H
   12 CONTINUE
      NSRCH=1
      CALL ONED
      IF(FO-FUN+EPS)50,28,28
C**** TEST FOR TERMINATION ****
   28 CALL EXTEST
      IF(KOUNT.LT.NWORK)IER=0
   29 IF(IER.GT.0)GOTO 56
      IF(AN.LE.0.)GOTO 100
   30 IF(NOR.EQ.3)GOTO35
C *** CALC DEL FOR STEWART ***
      PHI=FUN
      ABPHI=ABS(PHI)
      DO 33 I=1,N
      ALPHA=HINV(I)
      ABAL=ABS(ALPHA)
      GAM=DFDT(I)
      ABGAM=ABS(GAM)
      DELPH=DEL(I)
      IF(ABGAM.LT.(FUN*EPS))GOTO33
      ZET=T(I)
      ABZ=ABS(ZET)
      ETA=EPS
      DUM=ABS(GAM*ZET/PHI)*EPM
      IF(ETA.LT.DUM)ETA=DUM
      DUM=ABAL*ABPHI*ETA
      DELPH=ABPHI*ETA/ABAL
      IF((GAM**2).LT.DUM)GOTO31
      DELPH=2.*SQRT(DELPH)
      DELPH=DELPH*(1.-(ABAL*DELPH)/(3.*ABAL*DELPH+4.*ABGAM))
      GOTO32
   31 DELPH=2.*((DELPH*ABGAM/ABAL)**(1./3.))
      DELPH=DELPH*(1.-(2.*ABGAM)/(3.*ABAL*DELPH+4.*ABGAM))
   32 CONTINUE
      IF((ALPHA*GAM).LT.0.)DELPH=-DELPH
      KODER(I)=0
      IF((0.5*ABAL*ABS(DELPH)).LT.(0.05*ABGAM))GOTO33
      KODER(I)=1
```

```
      DUM=ABGAM/ABAL
      DELPH=-DUM+SQRT(DUM**2+200.*ABPHI*ETA/ABAL)
   33 DEL(I)=DELPH
      CALL DIFFUN
      GOTO 36
   35 CALL DERFUN
   36 CONTINUE
C       COMPUTE DIFFERENCE VECTORS OF ARGUMENT AND GRAD FROM
C       TWO CONSECUTIVE ITERATIONS
      DO 37 J=1,N
      K=N+J
      H(K)=DFDT(J)-H(K)
      K=N+K
   37 H(K)=T(J)-H(K)
      Z=0.
      DO 38 J=1,N
      K=N+J
      W=H(K)
      K=K+N
   38 Z=Z+W*H(K)
   39 IER=0
      IF(NOR.EQ.3)GOTO43
      BETA=0.
      RHO=0.
      DO41J=1,N
      K=N+J
      BETA=BETA+H(K)*H(J)
      RHO=RHO+(DFDT(J)-H(K))*H(J)
   41 CONTINUE
      C1=1./BETA
      C2=(1./AN-RHO*C1)
      DO42J=1,N
      K=N+J
   42 HINV(J)=HINV(J)+C1*((C2-2.)*H(K)+2.*DFDT(J))*H(K)
   43 CONTINUE
C          PREPARE UPDATING OF MATRIX H
      ALFA=0.
      DO 47 J=1,N
      K=J+N3
      W=0.
      DO 46 L=1,N
      KL=N+L
      W=W+H(KL)*H(K)
      IF(L-J)44,45,45
   44 K=K+N-L
      GO TO 46
   45 K=K+1
   46 CONTINUE
      K=N+J
      ALFA=ALFA+W*H(K)
   47 H(J)=W
```

```
C          REPEAT SEARCH IN DIRECTION OF STEEPEST DESCENT IF RESULTS
C          ARE NOT SATISFACTORY
      IF(Z*ALFA)48,1,48
   48 K=N31
      DO 49 L=1,N
      KL=N2+L
      DO 49 J=L,N
      NJ=N2+J
      H(K)=H(K)+H(KL)*H(NJ)/Z-H(L)*H(J)/ALFA
   49 K=K+1
      GO TO 5
C          END OF ITERATION LOOP
C          RESTORE OLD VALUES OF FUNCTION AND ARGUMENTS
   50 DO 51 J=1,N
      K=N2+J
   51 T(J)=H(K)
      FUN=FO
      DO 52 J=1,N
      K=N+J
   52 DFDT(J)=H(K)
C          REPEAT SEARCH IN DIRECTION OF STEEPEST DESCENT IF DERIVATIVE
C          FAILS TO BE SUFFICIENTLY SMALL
      IF(GNRM-EPS)55,55,53
C          TEST FOR REPEATED FAILURE OF ITERATION
   53 IF(IER)56,54,54
   54 IER=-1
      GOTO 1
   55 CONTINUE
   56 RETURN
      END


      SUBROUTINE MAP
C *** NOR=5 ***
C     THIS SUBROUTINE SETS UP THE LINEAR PROGRAMMING PROBLEM
COMMON CARDS:DATA,PRINT,OPT,WORK
      IER=0
      KOUNT=0
      IHE=1
      NW1=NWORK+1
      NCOL=NWORK
      NN=NWORK
      NNP1=NN+1
      IWORK=M*NLC
      IWORK2=2*IWORK
      NROW=IWORK2+NWORK
      NRP1=1+NROW
   50 CONTINUE
      DO 95 J=1,NWORK
   95 PO(J)=T(J)
   99 CONTINUE
      KOUNT=KOUNT+1
```

```
      FO=FUN
      CALL DERFUN
      DO 100 J=1,NWORK
      DO 100 L=1,NLC
      DO 100 K=1,M
      I=(L-1)*M+K
  100 A(I,J)=-DSDT(K,L,J)
      DO 105 J=1,NWORK
      DO 105 I=1,IWORK
  105 A(I+IWORK,J)=-A(I,J)
      DO 115 J=1,NWORK
      DO 110 I=1,NWORK
  110 A(IWORK2+I,J)=0.
  115 A(IWORK2+J,J)=1.
      DO 120 J=1,NWORK
  120 A(NRP1,J)=XL(J)
      A(NRP1,NNP1)=0.
      ONMA=ALPHA*(TMAX-TMIN)
      DO 135 I=1,NN
      TL(I)=TMIN
      TU(I)=TMAX
      RRR=T(I)-ONMA
      IF(RRR.GT.TL(I))TL(I)=RRR
      SSS=T(I)+ONMA
      IF(SSS.LT.TU(I))TU(I)=SSS
  135 CONTINUE
      DO 145 L=1,NLC
      DO 145 J=1,M
      JJ=M*(L-1)+J
      SUM=0.
      DO 140 K=1,NN
  140 SUM=SUM+DSDT(J,L,K)*(T(K)-TL(K))
      SUM=SUM-S(J,4,L)
      A(JJ,NNP1)=-SIGL-SUM
  145 A(JJ+IWORK,NNP1)=SIGA+SUM
      DO 150 I=1,NN
  150 A(IWORK2+I,NNP1)=TU(I)-TL(I)
      CALL PRMDUL
      DO 300 I=1,NWORK
      T(I)=H(I)
  300 T(I)=T(I)+TL(I)
      CALL FUNCT
      CALL EXTEST
      IF(IER.EQ.0)GOTO 50
  900 CONTINUE
      RETURN
      END


      SUBROUTINE MFD
C *** NOR=6 ***
COMMON CARDS:DATA,PRINT,OPT,WORK,Z
```

```
      NFAIL=0
      IER=0
      NP=NWORK
      NCOL=NP+1
      KOUNT=0
      IHE=1
      NP2=NP+2
      KU=2*(M*NLC+NP)
  100 KOUNT=KOUNT+1
      CALL DERFUN
      IF(KOUNT.GT.1)GOTO 125
      IF(IOPTS.GT.1)GOTO 125
      AM=0.
      DO 105 I=1,NP
  105 H(I)=-XL(I)
      FU=SIGA-TOL/2.
      FL=SIGL+TOL/2.
      CALL FSMOVE
      GOTO 100
  125 NROW=1
      IU=2*M*NLC
      DO 130 I=1,IU
      A(I,NCOL)=0.
  130 A(I,NP2)=0.
      GNRM=0.
      DO 150 I=1,NP
      A(1,NP2)=A(1,NP2)+XL(I)
      GNRM=GNRM+ABS(XL(I))
  150 A(1,I)=XL(I)
      A(1,NCOL)=GNRM
      FU=SIGA-TOL
      FL=SIGL+TOL
      TTOL=TOL
      TEST=1.E+50
      II=0
      DO 275 L=1,NLC
      DO 275 K=1,M
      DUM=S(K,4,L)
      II=1+II
      IACT=0
      IF(KM1(II)+KM2(II).GE.2)IACT=1
      KM2(II)=KM2(II)+KM1(II)
      KM1(II)=0
      IF(DUM.GE.FU)KM1(II)=1
      IF(KM1(II)+IACT.EQ.0)GOTO 225
      DDUM=SIGA-DUM
      IF(DDUM.LT.TTOL)TTOL=DDUM
      TTEST=TOL-DDUM
      NROW=1+NROW
      A(NROW,NCOL)=0.
      DO 200 I=1,NP
      A(NROW,NCOL)=A(NROW,NCOL)+ABS(DSDT(K,L,I))
```

```
      A(NROW,NP2)=A(NROW,NP2)+DSDT(K,L,I)
  200 A(NROW,I)=DSDT(K,L,I)
      A(NROW,NP2)=A(NROW,NP2)-TOL
      A(NROW,NCOL)=A(NROW,NCOL)*PSI
      TTEST=TTEST/A(NROW,NCOL)
      IF(TTEST.GE.0..AND.TTEST.LT.TEST)TEST=TTEST
  225 II=1+II
      IACT=0
      IF(KM1(II)+KM2(II).GE.2)IACT=1
      KM2(II)=KM2(II)+KM1(II)
      KM1(II)=0
      IF(DUM.LE.FL)KM1(II)=1
      IF(KM1(II)+IACT.EQ.0)GOTO 275
      DDUM=DUM-SIGL
      IF(DDUM.LT.TTOL)TTOL=DDUM
      TTEST=TOL-DDUM
      NROW=1+NROW
      A(NROW,NCOL)=0.
      DO 250 I=1,NP
      A(NROW,NCOL)=A(NROW,NCOL)+ABS(DSDT(K,L,I))
      A(NROW,NP2)=A(NROW,NP2)-DSDT(K,L,I)
  250 A(NROW,I)=-DSDT(K,L,I)
      A(NROW,NP2)=A(NROW,NP2)-TOL
      A(NROW,NCOL)=A(NROW,NCOL)*PSI
      TTEST=TTEST/A(NROW,NCOL)
      IF(TTEST.GE.0..AND.TTEST.LT.TEST)TEST=TTEST
  275 CONTINUE
      IF(NROW.EQ.1)TEST=0.
      IL=1+NROW
      IU=1+3*NP+IL
      DO 300 J=1,NCOL
      DO 300 I=IL,IU
  300 A(I,J)=0.
      FU=TMAX*(1.-TOL/SIGA)
      FL=TMIN*(1.+TOL/SIGA)
      DO 375 J=1,NP
      DUM=T(J)
      II=1+II
      IACT=0
      IF(KM1(II)+KM2(II).GE.2)IACT=1
      KM2(II)=KM2(II)+KM1(II)
      KM1(II)=0
      IF(DUM.GE.FU)KM1(II)=1
      IF(KM1(II)+IACT.EQ.0)GOTO 350
      NROW=1+NROW
      A(NROW,J)=1.
      A(NROW,NP2)=1.
  350 II=1+II
      IACT=0
      IF(KM1(II)+KM2(II).GE.2)IACT=1
      KM2(II)=KM2(II)+KM1(II)
      KM1(II)=0
```

```
      IF(DUM.LE.FL)KM1(II)=1
      IF(KM1(II)+IACT.EQ.0)GOTO 375
      NROW=1+NROW
      A(NROW,J)=-1.
      A(NROW,NP2)=-1.
  375 CONTINUE
      DO 400 J=1,NP
      NROW=1+NROW
      A(NROW,J)=1.
  400 A(NROW,NP2)=2.
      NRP1=NROW+1
      A(NRP1,NCOL)=-1.
      A(NRP1,NP2)=0.
C *** DIRECTION PROBLEM IS SET UP ***
      IISRCH=ISRCH
      ISRCH=10*NP
      CALL PRMDUL
      ISRCH=IISRCH
      IF(IER.GT.1)GOTO 462
      DO 460 I=1,NP
  460 H(I)=H(I)-1.
      CALL GETMA(AMAX,AMIN)
      IF(AMAX.EQ.0.)GOTO 462
      IF(ABS(H(NCOL)).LT.1.E-10)GOTO 999
      GOTO 475
  462 CONTINUE
      IER=0
      NFAIL=1+NFAIL
      IF(NFAIL.GE.10)GOTO 999
      DO 465 K=1,KU
      KM1(K)=0
      KM2(K)=0
  465 CONTINUE
      TOL=TOL/2.
      IF(NFAIL.GE.2)GOTO 125
      IF(TOL.LT.TTOL)TOL=TTOL
      GOTO 125
  475 CONTINUE
      AM=TEST/ABS(H(NCOL))
      NFAIL=0
      FU=SIGA-TOL/2.
      FL=SIGL+TOL/2.
      DO  495 I=1,NWORK
  495 PO(I)=T(I)
      FO=FUN
  500 CALL FSMOVE
      CALL EXTEST
      IF(IER.EQ.0)GOTO 100
  999 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE NEWT
C *** NOR=7 ***
C**** SUBROUTINE PERFORMS NEWTON-RAPHSON WITH ONE DIM. SEARCHES
COMMON CARDS:PRINT,OPT,WORK
    5 FORMAT(' HESS SINGULAR,RETURN')
      IER=0
      KOUNT=0
      IHE=2
  100 CONTINUE
      KOUNT=KOUNT+1
      FO=FUN
      DO 150 I=1,NWORK
  150 PO(I)=T(I)
      CALL DERFUN
      GNRM=0.
      DO 200 I=1,NWORK
      GNRM=GNRM+ABS(DFDT(I))
      H(I)=-DFDT(I)
      IF((GNRM-EPS).LE.0.)GOTO 999
      KH=1
      DO 300 J=1,NWORK
      DO 299 I=1,J
      HE(KH)=D2FDT2(I,J)
  298 KH=KH+1
  299 CONTINUE
  300 CONTINUE
  400 CALL GELS(H,HE,NWORK,1,EPS,IER,AUX)
      IF(KS.EQ.1)GOTO 500
      GO TO 600
  500 WRITE(6,5)
      GOTO999
  600 CONTINUE
      HNRM=0.
      DO 650 I=1,NWORK
      HNRM=HNRM+ABS(H(I))
  650 CONTINUE
      NSRCH=KOUNT
      CALL ONED
  700 CONTINUE
C**** CHECK FOR TERMINATION ****
      CALL EXTEST
      IF(IER.EQ.0)GOTO 100
  999 RETURN
      END


      SUBROUTINE QP
C *** NOR=8 ***
      RETURN
      END
```

```
      SUBROUTINE EXTEST
COMMON CARDS:PRINT,OPT
    1 FORMAT(' DESIGN NOT CHANGING MUCH')
    2 FORMAT(' FUNCTION NOT CHANGING MUCH')
    3 FORMAT(' NO CONVERGANCE AFTER',I5,'ITERNS')
      IER=0
      DO50I=1,NWORK
      QTEST=ABS(PO(I)-T(I))
      IF(QTEST.GT.(AL*T(I)))GOTO75
   50 CONTINUE
      WRITE(6,1)
      QTEST=ABS(FO-FUN)
      IF(QTEST.GT.(FUNL*FO))GOTO75
      WRITE(6,2)
      IER=1
      RETURN
   75 IF(KOUNT.GE.LIMIT)GOTO100
      RETURN
  100 WRITE(6,3)KOUNT
      IER=2
      RETURN
      END
```

COMMENT : SEARCH ALGORITHMS

```
      SUBROUTINE ONED
C *** THIS SUBROUTINE PERFORMS A ONE DIMENSIONAL SEARCH
C *** AT CONCLUSION IT YIELDS A NEW DESIGN AND NEW GRADIENTS
COMMON CARDS:PRINT,OPT,WORK
    1 FORMAT(' REGION CONVEX RETURN TO ALGORITHM')
    2 FORMAT(' CAN NOT FIND SECOND FEASIBLE POINT ')
    3 FORMAT(' CAN NOT FIND THIRD FEASIBLE POINT ')
    4 FORMAT(' CAN NOT FIND FOURTH FEASIBLE POINT ')
    8 FORMAT(' INTERVAL OF UNCERTAINTY BELOW ACCEPTABLE SO WE STOPPED')
    9 FORMAT(' SEARCH TERMINATED AFTER ',I3,' TRIES')
   19 FORMAT(' ABS(H)=0. RETURN **********')
   21 FORMAT(' REGION FLAT,SEARCH TERMINATED AT BEST POINT = ',E15.8)
   22 FORMAT(' LAST POINT FURTHEST FROM BEST POINT,SEARCH TERMINATED')
      NONED=NONED+1
      DQQ=0.
      DO105I=1,NWORK
      DQ=ABS(H(I))
      IF(DQ.LT.DQQ)GOTO105
      DQQ=DQ
  105 CONTINUE
      IF(DQQ.EQ.0.)GOTO995
      AOK=TACTN/DQQ
  110 CONTINUE
      KVEX=0
      ICNT=0
      IDIR=0
      IQF=0
      A1=0.
      A2=0.
      A3=0.
      A4=0.
      F2=0.
      F3=0.
      F4=0.
      F1=FUN
      FSAVE=FUN
C *** SAVE BEST ***
      AA=A1
      FF=F1
      AQ=A1
      FQ=F1
C *** ********* ***
  115 CALL GETMA(AMAX,AMIN)
      DO120I=1,NWORK
  120 TSAVE(I)=T(I)
      FSAVE=FUN
      AH=0.
```

```
      DO125J=1,NWORK
  125 AH=AH+H(J)*H(J)
      AH=SQRT(AH)
      DY=0.
C *** GET SECOND POINT ***
  200 CONTINUE
      IF(NOR.GT.1)GOTO205
  201 IF(NUSE.EQ.1)GOTO202
      A2=A1
      F2=F1
      A1=A0
      F1=F0
      GOTO295
  202 A2=5.*AOK
      GOTO220
  205 DO206J=1,NWORK
  206 DY=DY+H(J)*DFDT(J)
  212 IF(IP.LT.3)GOTO1012
 1012 CONTINUE
      IF(DY)215,216,213
  213 DO214J=1,NWORK
  214 H(J)=-H(J)
      DY=-DY
      ATEMP=-AMIN
      AMIN=-AMAX
      AMAX=ATEMP
  215 ALFA=(EST-FSAVE)/DY
      A2=10.*AOK
      IF(ALFA.GT.A2)A2=ALFA
      IF(A2.GT.1.)A2=1.
      IF(A2.LT.0.)A2=-A2
      GOTO220
  216 A2=1.
  220 CONTINUE
      AQ=A2
  221 IF(AQ.GE.AMAX)AQ=(A1+AMAX)/3.
  222 IF(AQ.LE.AMIN)AQ=(A1+AMIN)/3.
      DO226I=1,NWORK
  226 T(I)=TSAVE(I)+AQ*H(I)
      CALL FUNCT
      FQ=FUN
C *** IS 2ND POINT FEASIBLE
      CALL FEASQ(IFEAS)
      IF(IFEAS.EQ.0)GOTO295
      IF(AQ.GT.0.)AMAX=AQ
      IF(AQ.LT.0.)AMIN=AQ
      IF(ABS(A2).LT.AOK)GOTO 275
      ICNT=ICNT+1
      IF(ICNT.GT.10)GOTO275
      GOTO 220
  275 IDIR=IDIR+1
      IF(IDIR.GT.1)GOTO821
```

```
      DO280 J=1,NWORK
  280 H(J)=-H(J)
      ATEMP=-AMIN
      AMIN=-AMAX
      AMAX=ATEMP
      GOTO202
  295 ICNT=0
      A2=AQ
      F2=FQ
      IF(F2.GT.F1)GOTO 298
      IF(F1.GT.F2)AMIN=A1
      AA=A2
      FF=F2
      GOTO 300
  298 AMAX=A2
C *** A FEASIBLE 2ND POINT IS NOW FOUND
C *** GET THIRD POINT ***
  300 CONTINUE
      IF(AMAX-AMIN.LE.2.*AOK)GOTO 800
      FMIN=FF
  301 IF(NOR.GT.1)GOTO305
  302 IF(F2.LT.F1)GOTO303
      AQ=A1+2.*(A1-A2)
      FMIN=FF
      GOTO 310
  303 AQ=A2+2.*(A2-A1)
      FMIN=FF
      GOTO 310
  305 C1=(DY*(A1-A2)-(F1-F2))/((A1-A2)*(A1-A2))
      C2=DY-2.*C1*A1
C *** IS REGION CONCAVE:WILL A MAXIMUM BE PREDICTED? ***
  306 CT=EPM*(F2+F1)/((A2-A1)**2)
      IF(C1.GT.CT)GOTO307
      AQ=A2+3.*(A2-A1)
      FMIN=FF
      GOTO 310
  307 AQ=-C2/(2.*C1)
      FMIN=(C1*(AQ+A1)+C2)*(AQ-A1)+F1
  310 CONTINUE
  320 IF(AQ.GE.AMAX)AQ=(A1+A2+AMAX)/3.
  321 IF(AQ.LE.AMIN)AQ=(A1+A2+AMIN)/3.
      DO330I=1,NWORK
  330 T(I)=TSAVE(I)+AQ*H(I)
      CALL FUNCT
      FQ=FUN
      IF(A2.GT.A1)GOTO331
      ATEMP=A1
      FTEMP=F1
      A1=A2
      F1=F2
      A2=ATEMP
      F2=FTEMP
```

```
  331 IF(AQ.LT.A2.AND.AQ.GT.A1)GOTO 337
C *** CHECK FEASIBILITY ***
  335 CALL FEASQ(IFEAS)
      IF(IFEAS.EQ.0)GOTO337
      ICNT=ICNT+1
      IF(ICNT.GT.10)GOTO993
      IF(AQ.GT.A2)GOTO336
      AMIN=AQ
      GOTO 320
  336 AMAX=AQ
      GOTO 320
C *** REORDER POINTS ***
  337 ICNT=0
      A3=AQ
      F3=FQ
      IF(A3.GT.A2)GOTO339
      AT=A3
      FT=F3
      A3=A2
      F3=F2
      A2=AT
      F2=FT
      IF(A2.GT.A1)GOTO339
      AT=A2
      FT=F2
      A2=A1
      F2=F1
      A1=AT
      F1=FT
  339 CONTINUE
C *** A FEASIBLE 3RD POINT IS FOUND
C *** GET FOURTH POINT ***
  400 IF(FF.LE.FQ)GOTO 405
      AA=AQ
      FF=FQ
  405 CONTINUE
      IF(F3.GE.F2.AND.A3.LT.AMAX)AMAX=A3
      IF(F1.GE.F2.AND.A1.GT.AMIN)AMIN=A1
      IF(F1.GT.F2.AND.F2.GE.F3)GOTO 410
      IF(F1.LE.F2.AND.F2.LT.F3)GOTO 415
      GOTO 420
  410 IF(A2.GT.AMIN)AMIN=A2
      GOTO 420
  415 IF(A2.LT.AMAX)AMAX=A2
  420 CONTINUE
      IF(AMAX-AMIN.LE.2.*AOK)GOTO 800
C *** IS REGION CONVEX - WILL A MAXIMUM BE PREDICTED ? ***
      A31=A3-A1
      A21=A2-A1
      A32=A3-A2
      C1=(F1-F2)/(A21*A31)-(F2-F3)/(A32*A31)
      CT=EPM*(F3+F1)/(A3-A1)**2)
```

```
  425 IF(C1.GT.CT)GOTO 440
      IF(C1.LE.0.)GOTO 816
      IF(AA.LT.A3)GOTO 302
      A1=A2
      F1=F2
      A2=A3
      F2=F3
      GOTO 302
  440 CONTINUE
      KVEX=0
      ICNT=0
      IQF=IQF+1
      C2=(F2-F3)/(-A32)-C1*(A2+A3)
      AQ=-C2/(2.*C1)
      D2FDA2(NSRCH)=C1*2.
      FMIN=(C1*(AQ+A1)+C2)*(AQ-A1)+F1
  445 CONTINUE
  450 CONTINUE
  455 IF(AQ.GE.AMAX)AQ=(A2+A3+AMAX)/3.
  460 IF(AQ.LE.AMIN)AQ=(A1+A2+AMIN)/3.
      DO 465 I=1,NWORK
  465 T(I)=TSAVE(I)+AQ*H(I)
      CALL FUNCT
      FQ=FUN
C *** CHECK FEASIBILITY ***
  470 IF(AQ.LT.A3.AND.AQ.GT.A1)GOTO 485
  475 CALL FEASQ(IFEAS)
      IF(IFEAS.EQ.0)GOTO 485
      ICNT=ICNT+1
      IF(ICNT.GT.10)GOTO994
      IF(AQ.GT.A3)GOTO 480
      AMIN=AQ
      GOTO 450
  480 AMAX=AQ
      GOTO 450
C *** REORDER POINTS ***
  485 ICNT=0
      A4=AQ
      F4=FQ
      IF(A4.GT.A3)GOTO 490
      AT=A4
      FT=F4
      A4=A3
      F4=F3
      A3=AT
      F3=FT
      IF(A3.GT.A2)GOTO 490
      AT=A3
      FT=F3
      A3=A2
      F3=F2
      A2=AT
```

```
      F2=FT
      IF(A2.GT.A1)GOTO 490
      AT=A2
      FT=F2
      A2=A1
      F2=F1
      A1=AT
      F1=FT
  490 IF(IP.LT.2)GOTO 1025
 1025 CONTINUE
  495 CONTINUE
C *** FOURTH FEASIBLE POINT IS FOUND ***
C *** DISCARD ONE POINT ***
  500 CONTINUE
  505 IF(F2.LT.F1.AND.A1.GT.AMIN)AMIN=A1
  515 IF(F3.LT.F4.AND.A4.LT.AMAX)AMAX=A4
      IF(AQ.EQ.A1.AND.AA.EQ.A4)GOTO 815
      IF(AA.EQ.A1.AND.AQ.EQ.A4)GOTO 815
      IF(AQ.EQ.A4)GOTO525
      IF(AQ.EQ.A1)GOTO535
      IF(AA.EQ.A4)GOTO525
      IF(AA.EQ.A1)GOTO 535
      IF(F3.LT.F2)GOTO520
      IF(F1.GT.F2)GOTO525
      GOTO535
  520 IF(F4.GT.F3)GOTO535
  525 A1=A2
      F1=F2
      A2=A3
      F2=F3
      A3=A4
      F3=F4
  535 A4=0.
      F4=0.
  545 CONTINUE
      IF(AMAX-AMIN.LE.2.*AOK)GOTO 800
C *** IS MINIMUM BOUND ? ***
      IF(F2.GT.F1.OR.F2.GT.F3)GOTO 400
C *** TEST FOR TERMINATION OF SEARCH ***
  700 CONTINUE
  705 IF((A3-A1).LE.(2.*AOK))GOTO 800
      IF(A1.EQ.A2.OR.A2.EQ.A3)GOTO 800
      IF(IQF.GE.ISRCH)GOTO 805
      GOTO 400
C *** AN EXIT REQUIREMENT HAS BEEN FULFILLED ***
  800 IF(IP.LT.2)GOTO 820
      WRITE(6,8)
      GOTO 820
  805 IF(IP.LT.2)GOTO 820
      WRITE(6,9)IQF
      GOTO820
  810 IF(IP.LT.2)GOTO 820
```

```
      WRITE(6,21)AA
      GOTO820
  815 IF(IP.LT.2)GOTO820
      WRITE(6,22)
      GOTO820
  816 IF(IP.LT.2)GOTO 820
      WRITE(6,1)
  820 CONTINUE
      IF(FQ.LT.FF)GOTO830
  821 AQ=AA
      DO825 I=1,NWORK
  825 T(I)=TSAVE(I)+AQ*H(I)
      CALL FUNCT
      FQ=FUN
      CALL FEASQ(IFEAS)
  830 CONTINUE
      AN=AQ
      GOTO999
C *** IF WE ARRIVED HERE AN ERROR IS APPARENT AND PROBLEM TERMINATED
  992 WRITE(6,2)
      GOTO821
  993 WRITE(6,3)
      GOTO821
  994 WRITE(6,4)
      GOTO821
  995 WRITE(6,19)
      AN=0.
  999 RETURN
        END


      SUBROUTINE FSMOVE
COMMON CARDS:DATA,PRINT,OPT,WORK
      NP=NWORK
      NONED=NONED+1
      IQF=0
      I1=0
      I2=0
      I3=0
      I4=0
      IQ=1
      IF1=0
      IF2=0
      IF3=0
      IF4=0
      IFQ=0
      A1=0.
      A2=0.
      A3=0.
      A4=0.
      AQ=0.
      DQQ=0.
```

```
      DO 110 I=1,NP
      DQ=ABS(H(I))
  110 IF(DQ.GT.DQQ)DQQ=DQ
      IF(DQQ.EQ.0.)GOTO 910
      AOK=TACTN/DQQ
      DO 120 I=1,NP
  120 TT(I)=T(I)
C *** FIND MOVE TO NON-LINEAR CONSTRAINTS ***
C *** UPPER BOUND FROM LINEAR CONSTRAINTS ***
C *** THEN FORM DIRECTIONAL DERIVATIVES   ***
      CALL GETMA(AMAX,AMIN)
      IF(AMIN.LT.0.)AMIN=0.
      IF(AM.LT.AMAX)AMIN=AM
      DO 250 K=1,M
      DO 250 L=1,NLC
      SS(K,1,L)=S(K,4,L)
      DO 220 IS=2,4
  220 SS(K,IS,L)=0.
      DY=0.
      DO 225 I=1,NP
  225 DY=DY+DSDT(K,L,I)*H(I)
  250 SS(K,4,L)=DY
      I1=1
C *** LINEAR FIT ***
  300 A2=AMAX
      DO 325 K=1,M
      DO 325 L=1,NLC
      DY=SS(K,4,L)
      IF(ABS(DY).LT.1.E-25)GOTO 325
      F1=SS(K,1,L)
      IF(DY.GT.0.)AT=(FU-F1)/DY
      IF(DY.LT.0.)AT=(FL-F1)/DY
      IF(AT.GT.AOK.AND.AT.LT.A2)A2=AT
  325 CONTINUE
      IF(A2.LT.AMIN)A2=AMIN
      AQ=A2
      DO 350 I=1,NP
  350 T(I)=TT(I)+AQ*H(I)
      CALL FUNCT
      CALL FEASQ(IFEAS)
      DO 375 K=1,M
      DO 375 L=1,NLC
  375 SS(K,2,L)=S(K,4,L)
      I2=2
      IQ=2
      IF2=IFEAS
      IFQ=IFEAS
      IF(IFEAS.EQ.-1)GOTO 960
      IF(IFEAS.EQ.1.AND.AQ.LT.AMAX)AMAX=AQ
C *** QUADRATIC FIT TO DY,A1,A2 ***
  400 AT=0.
      A3=AMAX
```

```
      C1=1./(A1-A2)
      C2=0.
      C3=0.
      C4=A1+A2
      DUM=2.*EPS/((A2-A1)**2)
      DO 425 K=1,M
      DO 425 L=1,NLC
      F1=SS(K,1,L)
      F2=SS(K,2,L)
      DY=SS(K,4,L)
      QB=C1*(F1-F2)
      QA=C1*(DY-QB)
      QT=ABS(DUM*(F1+F2))
      IF(ABS(QA).LE.QT)GOTO 425
      QB=QB-QA*C4
      QC=F1-A1*(QA*A1+QB)
      AT=AMAX
      CALL ROOT(QA,QB,QC,AOK,AT,AMIN)
      IF(AT.LT.A3)A3=AT
  425 CONTINUE
      IF(A3.EQ.A2)A3=(A1+A2)/2.
  435 AQ=A3
      DO 450 I=1,NP
  450 T(I)=TT(I)+AQ*H(I)
      CALL FUNCT
      CALL FEASQ(IFEAS)
      DO 475 K=1,M
      DO 475 L=1,NLC
  475 SS(K,3,L)=S(K,4,L)
      I3=3
      IQ=3
      IF3=IFEAS
      IFQ=IFEAS
C *** ORDER PTS A1,A2,A3 ***
      IF(A3.GT.A2)GOTO 485
      AT=A3
      A3=A2
      A2=AT
      IT=I3
      I3=I2
      I2=IT
      IFT=IF3
      IF3=IF2
      IF2=IFT
  485 CONTINUE
      IF(IFEAS.EQ.-1)GOTO 960
      IF(IF1.EQ.0)AMIN=A1
      IF(IF2.EQ.0)AMIN=A2
      IF(IF3.EQ.0)AMIN=A3
      IF(IF3.EQ.1)AMAX=A3
      IF(IF2.EQ.1)AMAX=A2
C *** QUADRATIC FIT TO A1,A2,A3 ***
```

```
  500 AT=0.
      AA=A1
      IF(IF1.EQ.0.AND.A1.GT.AMIN)AMIN=A1
      IF(IF2.EQ.1)GOTO505
      AA=A2
      IF(A2.GT.AMIN)AMIN=A2
      IF(IF3.EQ.1)GOTO510
      AA=A3
      IF(A3.GT.AMIN)AMIN=A3
      GOTO 515
  505 IF(A2.LT.AMAX)AMAX=A2
      GOTO 515
  510 IF(A3.LT.AMAX)AMAX=A3
  515 CONTINUE
      A4=AMAX
      IQF=IQF+1
      C1=1./(A1-A2)
      C2=1./(A2-A3)
      C3=1./(A3-A1)
      C4=A1+A2
      DUM=2.*EPS/((A3-A1)**2)
      DO 525 K=1,M
      DO 525 L=1,NLC
      F1=SS(K,I1,L)
      F2=SS(K,I2,L)
      F3=SS(K,I3,L)
      QB=C1*(F1-F2)
      QA=C3*(C2*(F2-F3)-QB)
      QT=ABS(DUM*(F1+F3))
      IF(ABS(QA).LE.QT)GOTO 525
      QB=QB-C4*QA
      QC=F1-(QA*A1+QB)*A1
      AT=AMAX
      CALL ROOT(QA,QB,QC,AOK,AT,AMIN)
      IF(AT.LT.A4)A4=AT
  525 CONTINUE
      IF(A4.EQ.A3)A4=(A3+A2)/2.
      IF(A4.EQ.A2)A4=(A1+A2)/2.
  535 AQ=A4
      IQ=4
      DO 550 I=1,NP
  550 T(I)=TT(I)+AQ*H(I)
      CALL FUNCT
      CALL FEASQ(IFEAS)
      IF4=IFEAS
      IFQ=IFEAS
      I4=4
C *** ORDER PTS A1 A2 A3 A4 ***
      IF(A4.GT.A3)GOTO 585
      AT=A4
      A4=A3
      A3=AT
```

```
      IT=I4
      I4=I3
      I3=IT
      IFT=IF4
      IF4=IF3
      IF3=IFT
      IF(A3.GT.A2)GOTO 585
      AT=A3
      A3=A2
      A2=AT
      IT=I3
      I3=I2
      I2=IT
      IFT=IF3
      IF3=IF2
      IF2=IFT
      IF(A2.GT.A1)GOTO 585
      AT=A2
      A2=A1
      A1=AT
      IT=I2
      I2=I1
      I1=IT
      IFT=IF2
      IF2=IF1
      IF1=IFT
  585 CONTINUE
      IF(IFEAS.EQ.-1)GOTO 960
      IF(IFEAS.EQ.1.AND.AQ.LT.AMAX)AMAX=AQ
      IF(IFEAS.EQ.0.AND.AQ.GT.AMIN)AMIN=AQ
C *** TEST FOR TERMINATION
  600 CONTINUE
      IF(IQF.GE.ISRCH)GOTO 920
      IF(IF1.EQ.0)GOTO 625
      IF(A1.EQ.0.)GOTO 625
  625 IF(IF2.EQ.0)GOTO 635
      AT=A2-A1
      GOTO 655
  635 IF(IF3.EQ.0)GOTO 645
      AT=A3-A2
      GOTO 655
  645 IF(IF4.EQ.0)GOTO 700
      AT=A4-A3
  655 AT=ABS(AT)
      IF(AT.LE.AOK)GOTO 930
C *** NO TERMINATION CONDITION FULFILLED ***
C *** SELECT REDUNDANT POINT             ***
  700 IFT=0
      IF(IF1.EQ.IFT)IFT=1
      IDIS=I4
      ADIS=A4
      IF(IF2.EQ.IFT)GOTO 710
```

```
      IDIS=I1
      ADIS=A1
      IF(IQ.NE.I1)GOTO 720
      IDIS=I2
      ADIS=A2
      IF(IF3.NE.IFT)GOTO 720
      IDIS=I4
      ADIS=A4
      GOTO 720
  710 IF(IQ.NE.I4)GOTO 720
      IDIS=I3
      ADIS=A3
  720 CONTINUE
C *** UPDATE MATRIX + INDICES ***
      DO 725 K=1,M
      DO 725 L=1,NLC
  725 SS(K,IDIS,L)=S(K,4,L)
      IF(AQ.EQ.A1)I1=IDIS
      IF(AQ.EQ.A2)I2=IDIS
      IF(AQ.EQ.A3)I3=IDIS
      IF(AQ.EQ.A4)I4=IDIS
      IF(ADIS.EQ.A1)GOTO 735
      IF(ADIS.EQ.A2)GOTO 740
      IF(ADIS.EQ.A3)GOTO 745
      IF(ADIS.EQ.A4)GOTO 750
  735 A1=A2
      I1=I2
      IF1=IF2
  740 A2=A3
      I2=I3
      IF2=IF3
  745 A3=A4
      I3=I4
      IF3=IF4
  750 A4=0.
      I4=0
      IF4=0
C *** POINT DISCARDED,FIND NEW POINT ***
      GOTO 500
C *** ERROR OR TERMINATION MESSAGES ***
  910 CONTINUE
      WRITE(6,6)
      CALL EXIT
  920 IF(IP.LT.2)GOTO 940
      WRITE(6,1)
      GOTO 940
  930 IF(IP.LT.2)GOTO940
      WRITE(6,2)
      GOTO 940
  940 IF(AA.EQ.AQ)GOTO960
      IF(IFQ.EQ.0)GOTO960
      AQ=AA
```

```
      DO 950 I=1,NP
  950 T(I)=TT(I)+AQ*H(I)
      CALL FUNCT
  960 IF(IP.LT.2)GOTO980
  980 IF(IP.LT.1)GOTO 990
  990 RETURN
    1 FORMAT(' MAX = OF QUADRATIC FITS REACHED')
    2 FORMAT(' CONSTRAINT LIES IN INTERVAL LESS THAN RESOLUTION')
    6 FORMAT(' MAX COMPONENT OF H = 0.0     ')
      END


      SUBROUTINE PRMDUL
C *** A PRIMAL-DUAL LINEAR PROGRAMMING ALGORITHM ***
COMMON CARDS:PRINT,OPT,WORK
    3 FORMAT(' LP SOLUTION UNBOUNDED. EXECUTATION TERMINATED')
    4 FORMAT(' LP ALGORITHM ANTICIPATES LOOPING. EXECUTATION TERMINATED'
     X)
    8 FORMAT(' CYCLING PREVENTION ALGORITHM ERROR NO 1')
    9  FORMAT(' CYCLE PRENTION ALGORITHM ERROR NO 2')
   13 FORMAT(' CAN NOT FIND INITIAL FEASIBLE SOLUTION')
   19 FORMAT('0A PIVOT CAN NOT BE FOUND AFTER',I3,' ITERATIONS')
C     N        = NUMBER OF COLUMNS IN COEF MATRIX OF INEQUALITY EQS.
C     M        = NUMBER OF ROWS IN COEF MATRIX OF INEQUALITY EQS
C     A        = MATRIX CONTAINING COEFS,COSTS,RH SIDES,AND OF
C     IREM()     = ROW DESIGNATION OF ZERO B'S
      N=NCOL
      M=NROW
      NP1=N+1
      MP1=M+1
       NDUM=N+M
      ISAVE=0
      JSAVE=0
       DO98J=1,NDUM
   98  ICOEF(J)=J
      ICOUNT=0
   99  CONTINUE
      ICOUNT=ICOUNT+1
      ATEST=1.E+75
      ICK=0
      JCK=0
      CTEST=1.
      DO100J=1,N
      IF(A(MP1,J).GE.CTEST)GOTO100
      JCK=J
      CTEST=A(MP1,J)
  100 CONTINUE
      IF(CTEST.LT.0.)GOTO 101
      GOTO 161
  101 IF(ICOUNT.GE.ISRCH)GOTO 952
      ICTEST=0
```

```
C *** FIND A PIVOT FOR PRIMAL PROBLEM ***
      DO 102 I=1,M
      IF(A(I,JCK)*A(I,NP1).LT.0.)GOTO 103
      IF(A(I,JCK).GT.0.)GOTO 103
  102 CONTINUE
C     SOLUTION UNBOUNDED
      GOTO951
  103 CONTINUE
       DO111I=1,M
  111  IREM(I)=0
       K=0
      DO 114 I=1,M
      IF(A(I,JCK)*A(I,NP1))114,112,113
  112 IF(A(I,NP1).NE.0.)GOTO 114
      IF(A(I,JCK).EQ.0.)GOTO 114
       K=K+1
      IREM(K)=I
       ATEST=0.
       ICK=I
      GOTO 114
  113 CONTINUE
      ATEST1=A(I,NP1)/A(I,JCK)
      IF(ATEST1.GT.ATEST)GOTO 114
      ATEST=ATEST1
      ICK=I
  114 CONTINUE
      NLOOK=N
  118 CONTINUE
      IF(K.LT.2)GOTO 153
C     AT LEAST 2 B'S ZERO
      NLOOK=NLOOK+1
      NTEST=N+M+1
      IF(NLOOK.GE.NTEST)GOTO953
      DO 123 J=1,NDUM
      IF(ICOEF(J).EQ.NLOOK)GOTO 124
  123 CONTINUE
      GOTO954
  124 CONTINUE
      JLOOK=J
      IF(JLOOK.LE.N)GOTO 141
      IDUMY=JLOOK-N
      DO 131 I=1,MP1
  131 DUMMY(I)=0.
      DUMMY(IDUMY)=1.
      GOTO 143
  141 DO 142 I=1,MP1
  142 DUMMY(I)=A(I,JLOOK)
  143 CONTINUE
      KK=0
      ATEST=1.E+75
      DO 152 I=1,K
      II=IREM(I)
```

```
      IF(DUMMY(II).NE.0.)GOTO 151
      KK=KK+1
      IREM(KK)=II
      ATEST=0.
      ICK=II
      GOTO 152
  151 ATEST1=DUMMY(II)/A(II,JCK)
      IF(ATEST1.GT.ATEST)GOTO 152
      ATEST=ATEST1
      ICK=II
  152 CONTINUE
      K=KK
      GOTO118
  153 CONTINUE
C *** A(ICK,JCK) = PIVOT FOR PRIMAL SIMPLEX ***
      IF(ICK.EQ.ISAVE.AND.JCK.EQ.JSAVE)GOTO 952
      ISAVE=ICK
      JSAVE=JCK
      CALL SIMP(ICK,JCK)
      GOTO 99
C *** AN OPTIMAL SOLUTION HAS BEEN REACHED ***
  161 CONTINUE
C *** CHECK FEASIBILITY ***
      IICK=0
      JJCK=0
      TEST=0.
      DO 162 I=1,M
      IF(A(I,NP1).GT.TEST)GOTO 162
      TEST=A(I,NP1)
      IICK=I
  162 CONTINUE
      IF(TEST.GE.-1.E-8)GOTO 900
C *** FIND A PIVOT FOR DUAL PROBLEM ***
      ICOUNT=1+ICOUNT
      IF(ICOUNT.GE.ISRCH)GOTO 952
      ATEST=-1.E+70
      K=0
      DO 165 J=1,N
      IF(A(IICK,J)*A(MP1,J))164,163,165
  163 IF(A(MP1,J).NE.0.)GOTO 165
      IF(A(IICK,J).EQ.0.)GOTO 165
      K=1+K
      IREM(K)=J
      ATEST=0.
      JJCK=J
      GOTO 165
  164 ATEST1=A(MP1,J)/A(IICK,J)
      IF(ATEST1.LT.ATEST)GOTO 165
      ATEST=ATEST1
      JJCK=J
  165 CONTINUE
      IF(K.LT.2)GOTO 169
```

```
      ATEST=0.
      DO 167 J=1,N
      IF(A(MP1,J).NE.0.)GOTO 167
      DUM=ABS(A(IICK,J))
      IF(DUM.LT.ATEST)GOTO 167
      ATEST=DUM
      JJCK=J
  167 CONTINUE
  169 CONTINUE
C *** A(IICK,JJCK) = PIVOT FOR DUAL SIMPLEX ***
      IF(IICK*JJCK.EQ.0)GOTO 170
      IF(IICK.EQ.ISAVE.AND.JJCK.EQ.JSAVE)GOTO 952
      ISAVE=IICK
      JSAVE=JJCK
      CALL SIMP(IICK,JJCK)
      GOTO 161
  170 WRITE(6,19)ICOUNT
      IER=5
      GOTO 910
C *** A BASIC FEASIBLE OPTIMAL SOLUTION HAS BEEN REACHED
  900 CONTINUE
  910 IF(IP.LT.2)GOTO 1940
 1940 CONTINUE
  912 CONTINUE
      DO 915 J=1,N
  915 H(J)=0.
      DO 920 J=NP1,NDUM
      I=J-N
      K=ICOEF(J)
      IF(K.GT.N)GOTO 920
      H(K)=A(I,NP1)
  920 CONTINUE
      OF=-A(MP1,NP1)
       GOTO999
  951  WRITE(6,3)
      IER=3
      GOTO 910
  952  WRITE(6,4)
      IER=4
      GOTO 910
  953  WRITE(6,8)
      RETURN
  954  WRITE(6,9)
      RETURN
  955 WRITE(6,13)
  999  RETURN
       END


      SUBROUTINE SIMP(ICK,JCK)
C     THIS SUBROUTINE CHANGES TABLEAU
COMMON CARDS:PRINT,OPT,WORK
```

```
      N=NCOL
      M=NROW
      NP1=N+1
      MP1=M+1
C     ICK IS ROW DESIGNATION OF PIVOT
C     THE ICK+N COLUMN WILL LEAVE BASIS
      IREM1=ICOEF(JCK)
      ICOEF(JCK)=ICOEF(ICK+N)
      ICOEF(ICK+N)=IREM1
C     CHANGE TABLEAU
      A(ICK,JCK)=1./A(ICK,JCK)
      DO 105 J=1,NP1
      IF(J.EQ.JCK)GOTO 105
      A(ICK,J)=A(ICK,J)*A(ICK,JCK)
  105 CONTINUE
      DO 110 J=1,NP1
      IF(J.EQ.JCK)GOTO 110
      DO 110 I=1,MP1
      IF(I.EQ.ICK)GOTO 110
      A(I,J)=A(I,J)-A(I,JCK)*A(ICK,J)
  110 CONTINUE
      DO 115 I=1,MP1
      IF(I.EQ.ICK)GOTO 115
      A(I,JCK)=-A(I,JCK)*A(ICK,JCK)
  115 CONTINUE
      RETURN
      END


      SUBROUTINE ROOT(QA,QB,QC,AOK,AT,AMIN)
COMMON CARDS:OPT
      DUM2=.5/QA
      DUM1=QB*(-DUM2)
      DT=(AOK/DUM2)**2
      QT=(.01*(-QB))**2
      IF(DT.LT.QT)DT=QT
  100 DUM=QB*QB-4.*QA*(QC-FU)
      IF(DUM.GT.0.)GOTO 150
      IF(DUM**2.LT.DT**2)GOTO 120
      GOTO400
  120 IF(DUM1.GT.0.)AT=DUM1
      GOTO 400
  150 DUM=DUM2*SQRT(DUM)
      QT=DUM1+DUM
  200 IF(QT.GT.AMIN.AND.QT.LT.AT)AT=QT
      QT=DUM1-DUM
  300 IF(QT.GT.AMIN.AND.QT.LT.AT)AT=QT
  400 DUM=QB*QB-4.*QA*(QC-FL)
      IF(DUM.GT.0.)GOTO 450
      IF(DUM**2.LT.DT**2)GOTO 420
      GOTO700
  420 IF(DUM1.GT.0..AND.DUM1.LT.AT)AT=DUM1
```

```
      GOTO 700
  450 DUM=DUM2*SQRT(DUM)
      QT=DUM1+DUM
  500 IF(QT.GT.AMIN.AND.QT.LT.AT)AT=QT
      QT=DUM1-DUM
  600 IF(QT.GT.AMIN.AND.QT.LT.AT)AT=QT
  700 RETURN
      END


      SUBROUTINE FEASQ(IFEAS)
C     THIS SUBROUTINE TELLS WHETHER DESIGN IS FEASIBLE
COMMON CARDS:DATA,PRINT,OPT
    1 FORMAT(' IFEAS = ',I2,'  - CONSTRAINT TIGHT ')
    2 FORMAT(' IFEAS = ',I2,'  - DESIGN FEASIBLE ')
    3 FORMAT(' IFEAS = ',I2,'  - DESIGN UNFEASIBLE ')
      IFEAS=0
      DO 100 L=1,NLC
      DO 100 K=1,M
      TEST=S(K,4,L)
      TTEST=TEST-SIGL
      TEST=SIGA-TEST
      IF(TTEST.LT.TEST)TEST=TTEST
      IF(TEST.GT.TOL)GOTO 100
      IF(TEST.LT.-TOL)GOTO 200
      IF(NOR.EQ.5)GOTO 100
      IF(TEST.LT.0.)GOTO 200
      IF(NOR.NE.6)GOTO 100
      IFEAS=-1
  100 CONTINUE
      GOTO 300
  200 IFEAS=1
  300 IF(IP.LT.2)GOTO 400
      IF(IFEAS)325,350,375
  325 WRITE(6,1)IFEAS
      GOTO 400
  350 WRITE(6,2)IFEAS
      GOTO 400
  375 WRITE(6,3)IFEAS
  400 CONTINUE
      RETURN
      END


      SUBROUTINE GETMA(AMAX,AMIN)
C     THIS SUBROUTINE FINDS MAXIMUM VALUE OF A ALLOWABLE SO AS NOT TO
C     HAVE T .GE. TMAX OR T .LE. TMIN
COMMON CARDS:PRINT,OPT
    1 FORMAT(//' DIRECTION OF TRAVEL IS DETERMINED AS'//4X,1HI,6X,
     S9HDIRECTION/)
    2 FORMAT(I5,E15.4)
    3 FORMAT(//' MAXIMUM MOVE IN DIRECTION = ',E15.4,' MINIMUM = ',
```

```
     XE15.4)
      K=0
      DO100J=1,NWORK
      IF(H(J).GT.0.)GOTO105
      IF(H(J).LT.0.)GOTO110
      GOTO100
  105 CONTINUE
      AMAXT=(TMAX-T(J))/H(J)
      AMINL=(TMIN-T(J))/H(J)
      GOTO90
  110 CONTINUE
      AMAXT=(TMIN-T(J))/H(J)
      AMINL=(TMAX-T(J))/H(J)
   90 CONTINUE
      K=K+1
      IF(K.GE.2)GOTO91
      AMAX=AMAXT
      AMIN=AMINL
      GOTO100
   91 CONTINUE
      IF(AMAXT.GE.AMAX)GOTO 92
      AMAX=AMAXT
   92 IF(AMINL.LT.AMIN)GOTO 100
      AMIN=AMINL
  100 CONTINUE
      IF(IP.LT.1)GOTO200
      WRITE(6,1)
      WRITE(6,2)(I,H(I),I=1,NWORK)
      WRITE(6,3)AMAX,AMIN
  200 CONTINUE
      RETURN
      END
```

COMMENT : FUNCTION AND DERIVATIVE ALGORITHMS

```
      SUBROUTINE FUNCT
COMMON CARDS:DATA,PRINT,TIME,OPT
      CALL TIMER(DATE,TIME,VIRT,TOTAL)
      OPTIM=OPTIM+VIRT
      NFE=NFE+1
  300 CONTINUE
      CALL SOLVE
      IF(NOR.EQ.5.OR.NOR.EQ.6)GOTO150
      FUN1=0.
      FUN2=0.
      FUN3=0.
      DO100I=1,NWORK
      FUN1=FUN1+T(I)*XL(I)
      FUN2=FUN2+1./(TMAX-T(I))+1./(T(I)-TMIN)
  100 CONTINUE
      FUN2=(TMAX-TMIN)*FUN2*RP
      FUN3=0.
      DO110LC=1,NLC
      DO110I=1,M
      DUM=ABS(SIGA-S(I,4,LC))
      IF(DUM.LT.1.E-50)GOTO200
      DUM=ABS(S(I,4,LC)-SIGL)
      IF(DUM.LT.1.E-50)GOTO200
  110 FUN3=FUN3+1./(SIGA-S(I,4,LC))+1./(S(I,4,LC)-SIGL)
      FUN3=(SIGA-SIGL)*FUN3*RP
      FUN=FUN1+FUN2+FUN3
  112 CONTINUE
      GOTO175
  150 FUN=0.
      DO 155 I=1,NWORK
  155 FUN=FUN+T(I)*XL(I)
  175 CONTINUE
  200 CONTINUE
      CALL TIMER(DATE,TIME,VIRT,TOTAL)
      FUNTIM=FUNTIM+VIRT
      RETURN
      END


      SUBROUTINE DERFUN
COMMON CARDS:DATA,PRINT,TIME,OPT,WORK
      CALL TIMER(DATE,TIME,VIRT,TOTAL)
      OPTIM=OPTIM+VIRT
      NGE=NGE+1
      NN=N+N
      INLC=NLC
      DO 60 LC=1,NLC
      DO 50 IS=1,4
```

```
      DO 50 K=1,M
      SS(K,IS,LC)=S(K,IS,LC)
   50 CONTINUE
      DO 60 K =1,NN
      Q(K,LC)=P(K,LC)
      R(K,LC)=F(K,LC)
   60 CONTINUE
      DO 100  I=1,NWORK
      DFDT(I)=0.0
      DO 100 J=I,NWORK
      D2FDT2(I,J)=0.0
  100 CONTINUE
  105 CONTINUE
      NLC =1
      DO 7010 LC=1,INLC
      IGH=1
      DO 2000 I=1,NWORK
      DO 975 K=1,NN
  975 F(K,1)=0.0
      J=I
      IF(ISITP.EQ.2)GOTO1001
      DO 1000  K=1,M
      CALL EFFLD(I,J,K,LC)
 1000 CONTINUE
      GOTO1003
 1001 K=0
 1002 CONTINUE
      CALL EFFLD(I,J,K,LC)
 1003 CONTINUE
      CALL CHOS
      CALL FIXU
      DO 1050 L=1,NN
      DUDT(L,I)=P(L,1)
 1050 CONTINUE
 1061 CALL GETS
      IF(ISITP.EQ.2)GOTO1201
      DO 1200 K=1,M
      S(K,4,1)=(SS(K,3,LC)*6.*S(K,3,1)+SS(K,2,LC)*(2.*S(K,2,1)-S(K,1,1))
     C+SS(K,1,LC)*(2.*S(K,1,1)-S(K,2,1)))/(SS(K,4,LC)*2.)
 1200 CONTINUE
 1201 CONTINUE
      IF(NOR.NE.5.AND.NOR.NE.6)GOTO 1225
      DO 1220 K=1,M
 1220 DSDT(K,LC,I)=S(K,4,1)
      GOTO 2000
 1225 CONTINUE
      DO 1251 K=1,M
      DO 1250 IS=1,4
      DSDT(K,IS,I)=S(K,IS,1)
 1250 CONTINUE
 1251 CONTINUE
```

```
1253 CONTINUE
     DO 1500 K=1,M
     DUM=1./(SIGA-SS(K,4,LC))**2-1./(SS(K,4,LC)-SIGL)**2
     DFDT(I)=DFDT(I)+DSDT(K,4,I)*DUM
1499 CONTINUE
1500 CONTINUE
1999 CONTINUE
2000 CONTINUE
2001 CONTINUE
     IF(IHE.EQ.1)GOTO7007
     IF(NOR.EQ.5.OR.NOR.EQ.6)GOTO7007
     IF(IHE.EQ.2)GOTO3000
     GOTO8110
3000 IGH=2
3001 CONTINUE
     DO 7005 I=1,NWORK
     DO 7000 J=I,NWORK
4000 CONTINUE
     DO 4025 K=1,NN
4025 F(K,1)=0.0
     IF(ISITP.EQ.2)GOTO5001
     DO 5000 K=1,M
     CALL EFFLD(I,J,K,LC)
     CALL EFFLD(J,I,K,LC)
5000 CONTINUE
     GOTO5004
5001 K=0
5002 CONTINUE
     CALL EFFLD(I,J,K,LC)
5003 CONTINUE
     CALL EFFLD(J,I,K,LC)
5004 CONTINUE
5010 CONTINUE
     CALL CHOS
     CALL FIXU
     CALL GETS
     IF(ISITP.EQ.2)GOTO6001
     DO 6000 K=1,M
     S(K,4,1)=(SS(K,1,LC)*(2.*S(K,1,1)-S(K,2,1))+SS(K,2,LC)*(2.*S(K,2,1
    C)-S(K,1,1))+SS(K,3,LC)*6.*S(K,3,1)+DSDT(K,1,I)*(2.*DSDT(K,1,J)-DSD
    CT(K,2,J))+DSDT(K,2,I)*(2.*DSDT(K,2,J)-DSDT(K,1,J))+DSDT(K,3,I)*6.*
    CDSDT(K,3,J)-DSDT(K,4,I)*2.*DSDT(K,4,J))/(2.*SS(K,4,LC))
6000 CONTINUE
6001 CONTINUE
6986 CONTINUE
     DO 6990  K=1,M
     D2FDT2(I,J)=D2FDT2(I,J)+S(K,4,1)*(1./((SIGA-SS(K,4,LC))**2)-1./((SS
    C(K,4,LC)-SIGL)**2))+2.*DSDT(K,4,I)*DSDT(K,4,J)*(1./((SIGA-SS(K,4,
    CLC))**3)+1./((SS(K,4,LC)-SIGL)**3))
6990 CONTINUE
7000 CONTINUE
7005 CONTINUE
```

```
7007 CONTINUE
7010 CONTINUE
     IF(NOR.EQ.5.OR.NOR.EQ.6)GOTO8000
     DO 7050 I=1,NWORK
     DUM=(SIGA-SIGL)*DFDT(I)
     DFDT(I)=DUM
     DUM=1./(TMAX-T(I))**2-1./(T(I)-TMIN)**2
     DUM=(TMAX-TMIN)*DUM
     DUM=DFDT(I)+DUM
     DUM=RP*DUM
     DFDT(I)=DUM+XL(I)
7050 CONTINUE
7051 CONTINUE
     IF(IHE.EQ.1)GOTO8000
7100 CONTINUE
     DO 7255 I=1,NWORK
     DO 7250 J=I,NWORK
     D2FDT2(I,J)=(SIGA-SIGL)*D2FDT2(I,J)
7125 CONTINUE
     IF(I-J)7200,7150,7200
7150 D2FDT2(I,J)=D2FDT2(I,J)+2.*(TMAX-TMIN)*((1./(TMAX-T(I))**3)+
    C              (1./(T(I)-TMIN))**3)
7200 CONTINUE
7210 CONTINUE
     D2FDT2(I,J)=RP*D2FDT2(I,J)
7211 CONTINUE
     D2FDT2(J,I)=D2FDT2(I,J)
7250 CONTINUE
7255 CONTINUE
8000 CONTINUE
     NLC=INLC
     DO 8100 LC=1,NLC
     DO 8050 K=1,NN
     F(K,LC)=R(K,LC)
     P(K,LC)=Q(K,LC)
8050 CONTINUE
     DO 8100 IS=1,4
     DO 8100 K=1,M
     S(K,IS,LC)=SS(K,IS,LC)
8100 CONTINUE
     CALL TIMER(DATE,TIME,VIRT,TOTAL)
     DERTIM=DERTIM+VIRT
     RETURN
8110 CALL EXIT
     END


     SUBROUTINE EFFLD(I,J,K,LC)
COMMON CARDS:DATA,PRINT,OPT,WORK
   4 FORMAT(' ERROR :IGH=',I2)
 100 IF(IGH.NE.1)GOTO200
     DO 105 L=1,NK
```

```
      P(L,1)=Q(L,LC)
      GOTO 210
  200 IF(IGH.NE.2)GOTO207
  201 DO 205 L=1,NK
      P(L,1)=DUDT(L,J)
  205 CONTINUE
  105 CONTINUE
      GOTO 210
  207 WRITE(6,4)IGH
  210 CONTINUE
  213 CONTINUE
      IF(ISITP.EQ.2)GOTO280
      IF(NOD1(K).EQ.I)GO TO 225
      IF(NOD3(K).EQ.I)GO TO 225
      IF(NOD2(K).EQ.I)GO TO 225
      GOTO275
  225 IF(IP.LT.4)GOTO226
  226 CONTINUE
      N1=NOD1(K)+NOD1(K)-1
      N2=N1+1
      N3=NOD2(K)+NOD2(K)-1
      N4=N3+1
      N5=NOD3(K)+NOD3(K)-1
      N6=N5+1
      M1=21*(K-1)+1
      M2=M1+1
      M3=M2+1
      M4=M3+1
      M5=M4+1
      M6=M5+1
      M7=M6+1
      M8=M7+1
      M9=M8+1
      M10=M9+1
      M11=M10+1
      M12=M11+1
      M13=M12+1
      M14=M13+1
      M15=M14+1
      M16=M15+1
      M17=M16+1
      M18=M17+1
      M19=M18+1
      M20=M19+1
      M21=M20+1
      D1=-(AK(M1)*P(N1,1)+AK(M2)*P(N2,1)+AK(M3)*P(N3,1)
     1    +AK(M4)*P(N4,1)+AK(M5)*P(N5,1)+AK(M6)*P(N6,1))/3
      D2=-(AK(M2)*P(N1,1)+AK(M7)*P(N2,1)+AK(M8)*P(N3,1)
     1    +AK(M9)*P(N4,1)+AK(M10)*P(N5,1)+AK(M11)*P(N6,1))/3
      D3=-(AK(M3)*P(N1,1)+AK(M8)*P(N2,1)+AK(M12)*P(N3,1)
     1    +AK(M13)*P(N4,1)+AK(M14)*P(N5,1)+AK(M15)*P(N6,1))/3
      D4=-(AK(M4)*P(N1,1)+AK(M9)*P(N2,1)+AK(M13)*P(N3,1)
```

```
   1     +AK(M16)*P(N4,1)+AK(M17)*P(N5,1)+AK(M18)*P(N6,1))/3
      D5=-(AK(M5)*P(N1,1)+AK(M10)*P(N2,1)+AK(M14)*P(N3,1)
   1     +AK(M17)*P(N4,1)+AK(M19)*P(N5,1)+AK(M20)*P(N6,1))/3
      D6=-(AK(M6)*P(N1,1)+AK(M11)*P(N2,1)+AK(M15)*P(N3,1)
   1     +AK(M18)*P(N4,1)+AK(M20)*P(N5,1)+AK(M21)*P(N6,1))/3
      F(N1,1)=F(N1,1)+D1
      F(N2,1)=F(N2,1)+D2
      F(N3,1)=F(N3,1)+D3
      F(N4,1)=F(N4,1)+D4
      F(N5,1)=F(N5,1)+D5
      F(N6,1)=F(N6,1)+D6
      GOTO284
  275 IF(IP.LT.4)GOTO276
  276 CONTINUE
      GOTO284
  280 CONTINUE
      N1=NOD1(I)+NOD1(I)-1
      N2=N1+1
      N3=NOD2(I)+NOD2(I)-1
      N4=N3+1
      M1=10*(I-1)+1
      M2=M1+1
      M3=M2+1
      M4=M3+1
      M5=M4+1
      M6=M5+1
      M7=M6+1
      M8=M7+1
      M9=M8+1
      M10=M9+1
      D1=-(AK(M1)*P(N1,1)+AK(M2)*P(N2,1)+AK(M3)*P(N3,1)+AK(M4)*
     XP(N4,1))
      D2=-(AK(M2)*P(N1,1)+AK(M5)*P(N2,1)+AK(M6)*P(N3,1)+AK(M7)*
     XP(N4,1))
      D3=-(AK(M3)*P(N1,1)+AK(M6)*P(N2,1)+AK(M8)*P(N3,1)+AK(M9)*
     XP(N4,1))
      D4=-(AK(M4)*P(N1,1)+AK(M7)*P(N2,1)+AK(M9)*P(N3,1)+AK(M10)*
     XP(N4,1))
      F(N1,1)=F(N1,1)+D1
      F(N2,1)=F(N2,1)+D2
      F(N3,1)=F(N3,1)+D3
      F(N4,1)=F(N4,1)+D4
  284 CONTINUE
  286 CONTINUE
      RETURN
      END


      SUBROUTINE DIFFUN
COMMON CARDS:DATA,PRINT,OPT,WORK
      FOO=FUN
      DO 105 I=1,NWORK
```

```
 105 TT(I)=T(I)
     DO 110 I=1,NWORK
     T(I)=TT(I)+DEL(I)
     CALL FUNCT
     FPO(I)=FUN
     DFDT(I)=(FUN-FOO)/DEL(I)
     IF(KODER(I).EQ.0)GOTO110
     T(I)=TT(I)-DEL(I)
     CALL FUNCT
     DFDT(I)=(FPO(I)-FUN)/(2.*DEL(I))
 110 T(I)=TT(I)
1015 CONTINUE
     IF(IHE.EQ.1)GOTO140
 115 DO 130 I=1,NWORK
     T(I)=TT(I)+DEL(I)
     JU=I-1
     IF(JU.EQ.0)GOTO121
     DO 120 J=1,JU
     T(J)=TT(J)+DEL(J)
     CALL FUNCT
     FPP=FUN
     D2FDT2(J,I)=(FPP-FPO(I)-FPO(J)+FOO)
     DUM=DEL(I)*DEL(J)
     D2FDT2(J,I)=D2FDT2(J,I)/DUM
 120 T(J)=TT(J)
 121 T(I)=TT(I)-DEL(I)
     CALL FUNCT
     DUM=FPO(I)-FOO-FOO+FUN
     D2FDT2(I,I)=DUM/(DEL(I)**2)
 130 T(I)=TT(I)
 140 FUN=FOO
     RETURN
     END
```

COMMENT : STRUCTURAL ANALYSIS ALGORITHMS

```
      SUBROUTINE SOLVE
      CALL MERGE
      CALL DCOP
      CALL CHOS
      CALL FIXU
      CALL GETS
      RETURN
      END

      SUBROUTINE INIT
      CALL DAT
      CALL GIBW
      CALL INWK
      CALL GETAK
      RETURN
      END


      SUBROUTINE DAT
C     THIS SUBROUTINE READS DATA
COMMON CARDS:DATA,PRINT,OPT
    1 FORMAT(1H1)
    2 FORMAT(I3)
    3 FORMAT(19H NUMBER OF NODES = ,I3//5H NODE,14X,1HX,14X,1HY,14X,1HT/
     C)
    4 FORMAT(2F15.4)
 5    FORMAT(1X,I4,3F15.4)
    6 FORMAT(21H NUMBER OF MEMBERS = ,I3//5H  MEM,3X,2HN1,3X,2HN2,3X,3H
     XN3,4X,11HAREA IF ROD/)
    7 FORMAT(4I3,F15.4)
 8    FORMAT(1X,I4,3I5)
    9 FORMAT(33H NUMBER OF BOUNDARY CONDITIONS = ,I3//5H NODE,3X,2HBC)
 10   FORMAT(1X,I4,I5)
   12 FORMAT(F15.4)
   13 FORMAT(14H PRINT CODE = ,I3)
   16 FORMAT(19H NUMBER OF LOADS = ,I3//5H NODE,5H CODE,9X,6HAMOUNT/)
 17   FORMAT(1X,I4,I5,F15.4)
   18 FORMAT(28H SIZE OF STIFFNESS MATRIX = ,I3)
   19 FORMAT(I5)
   21 FORMAT(12H MOD ELAS = ,E15.4/18H POISSONS RATIO = ,E15.4)
   30 FORMAT(I5,4E15.4)
   31 FORMAT(21H MAXIMUM THICKNESS = ,E15.4/21H MINIMUM THICKNESS = ,
     XE15.4/' ALLOWABLE STRESS IN TENSION = ',E15.4/' ALLOWABLE STRESS I
     XN COMPRESSION = ',E15.4/'0NUMBER OF ITERATIONS FOR WHICH RP LOWERE
     XD',16X,'= ',I5)
   32 FORMAT(' UOA TERMINATES WHEN (WTI-WTIM1)/WTI LESS THAN WTEST',6X,'
     X=',E15.4)
   33 FORMAT(24H NUMBER OF LOAD CASES = ,I3)
   34 FORMAT(E15.4)
   35 FORMAT(11H DENSITY = ,E15.4)
```

```
 36 FORMAT(13H LOAD CASE = ,I3)
 38 FORMAT(I5,F15.4)
 39 FORMAT(' WE ARE DOING PLATE PROBLEM IF ISITP IS 1, ROD PROBLEM IF
   X2, ISITP = ',I3)
 40 FORMAT(1X,I4,2I5,5X,F15.4)
 41 FORMAT(' RESOLUTION FOR DESIGN  VARIABLES IS TACTN',16X,' =',
   XE15.4)
 42 FORMAT('SUBROUTINE DAT')
 44 FORMAT('0UNCONSTRAINED OPTIMIZATION ALGORITHM NUMBER',13X,'=',I5)
 45 FORMAT(' UOA TERMINATES WHEN AL DESIGN CHANGES LESS THAN AL',7X,
   X'=',E15.4/21X,'AND FUN CHANGES LESS THAN FUNL',7X,'=',E15.4)
    READ(5,2)ISITP
    WRITE(6,39)ISITP
    WRITE(6,1)
    READ(5,2) N
    WRITE(6,3) N
    DO 100 I=1,N
    READ(5,5) I,X(I),Y(I),T(I)
100 WRITE(6,5) I,X(I),Y(I),T(I)
    WRITE(6,1)
    READ(5,2) M
    WRITE(6,6) M
    DO 110 I=1,M
    READ(5,7) ID,NOD1(I),NOD2(I),NOD3(I),DUMMY
    IF(ISITP.EQ.1)GOTO105
    T(I)=DUMMY
    WRITE(6,40)I,NOD1(I),NOD2(I),T(I)
    GOTO110
105 CONTINUE
    WRITE(6,8) I,NOD1(I),NOD2(I),NOD3(I)
110 CONTINUE
    WRITE(6,1)
    READ(5,2) NB
    WRITE(6,9) NB
    DO 120 I=1,NB
    READ(5,19)  IB(I)
120 WRITE(6,10) I,IB(I)
    NK=2*N
    WRITE(6,1)
    READ(5,2)NLC
    WRITE(6,33)NLC
    DO 130 I=1,NK
    DO130LC=1,5
    F(I,LC)=0.
130 P(I,LC)=0.
    DO141LC=1,NLC
    WRITE(6,1)
    WRITE(6,36)LC
    READ(5,2) NL
    WRITE(6,16) NL
    DO140 I=1,NL
    READ(5,17) IN,IC,AMNT
```

```
      WRITE(6,17) IN,IC,AMNT
      ID1=2*(IN-1)+IC
  140 F(ID1,LC)=AMNT
  141 CONTINUE
      WRITE(6,1)
      READ(5,2) IP
      WRITE(6,13) IP
      WRITE(6,18) NK
      READ(5,4) EE,EENU
      WRITE(6,21) EE,EENU
      READ(5,12)RHO
      WRITE(6,35)RHO
      READ(5,30)NRPV,TMAX,TMIN,SIGA,SIGL
      WRITE(6,31)TMAX,TMIN,SIGA,SIGL,NRPV
      READ(5,30)LIMIT,AL,FUNL,TACTN,WTEST
      WRITE(6,32)WTEST
      WRITE(6,45)AL,FUNL
      WRITE(6,41)TACTN
      READ(5,2)NOR
      WRITE(6,44)NOR
      RETURN
      END


      SUBROUTINE GIBW
COMMON CARDS:DATA
 1    FORMAT(14H BAND WIDTH = ,I3)
      IBW=0
      IF(ISITP.EQ.2)GOTO200
      DO 100 I=1,M
      IDM=2*(NOD3(I)-NOD1(I)+1)
      IF(IDM.LE.IBW) GO TO 100
      IBW=IDM
 100   CONTINUE
      GOTO300
  200 CONTINUE
      DO250I=1,M
      IDM=2*(NOD2(I)-NOD1(I)+1)
      IF(IDM.LE.IBW)GOTO250
      IBW=IDM
  250 CONTINUE
  300 CONTINUE
      WRITE(6,1) IBW
      RETURN
      END


      SUBROUTINE INWK
COMMON CARDS:DATA,PRINT
      NT=0
      DO 100 I=1,NK
      ID1=I+IBW-1
```

```
      IF(ID1.GT.NK) GO TO 60
      NTIM(I)=IBW
      GO TO 100
 60   NTIM(I)=NK-I+1
 100  NT=NT+NTIM(I)
      ISUM(1)=1
      DO 200 I=2,NK
      IM1=I-1
 200  ISUM(I)=ISUM(IM1)+NTIM(IM1)
 1000 CONTINUE
      RETURN
      END


      SUBROUTINE GETAK
COMMON CARDS:DATA,PRINT,WORK
      DO90 I=1,N
   90 XL(I)=0.
      IF(ISITP.EQ.2)GOTO2000
      IDUM=0
      DO1000MEM=1,M
      N1=NOD1(MEM)
      N2=NOD2(MEM)
      N3=NOD3(MEM)
      X1=X(N1)
      X2=X(N2)
      X3=X(N3)
      Y1=Y(N1)
      Y2=Y(N2)
      Y3=Y(N3)
      X31=X3-X1
      Y31=Y3-Y1
      X32=X3-X2
      Y32=Y3-Y2
      X21=X2-X1
      Y21=Y2-Y1
      A123=.5*(X32*Y21-X21*Y32)
      A123=ABS(A123)
      C1=EE   /(4.*A123*(1.-EENU*EENU))
      C2=EE   /(8.*A123*(1.+EENU))
      V=EENU
      EKL(1)=C1*Y32*Y32+C2*X32*X32
      EKL(2)=-C1*V*Y32*X32-C2*X32*Y32
      EKL(3)=-C1*Y32*Y31-C2*X32*X31
      EKL(4)=C1*V*Y32*X31+C2*X32*Y31
      EKL(5)=C1*Y32*Y21+C2*X32*X21
      EKL(6)=-C1*V*Y32*X21-C2*X32*Y21
      EKL(7)=C1*X32*X32+C2*Y32*Y32
      EKL(8)=C1*V*X32*Y31+C2*Y32*X31
      EKL(9)=-C1*X32*X31-C2*Y32*Y31
      EKL(10)=-C1*V*X32*Y21-C2*Y32*X21
      EKL(11)=C1*X32*X21+C2*Y32*Y21
```

```
      EKL(12)=C1*Y31*Y31+C2*X31*X31
      EKL(13)=-C1*V*Y31*X31-C2*X31*Y31
      EKL(14)=-C1*Y31*Y21-C2*X31*X21
      EKL(15)=C1*V*Y31*X21+C2*X31*Y21
      EKL(16)=C1*X31*X31+C2*Y31*Y31
      EKL(17)=C1*V*X31*Y21+C2*Y31*X21
      EKL(18)=-C1*X31*X21-C2*Y31*Y21
      EKL(19)=C1*Y21*Y21+C2*X21*X21
      EKL(20)=-C1*V*Y21*X21-C2*X21*Y21
      EKL(21)=C1*X21*X21+C2*Y21*Y21
      DO300J=1,21
      IDUM=IDUM+1
  300 AK(IDUM)=EKL(J)
      A123=A123/3.
      XL(N1)=XL(N1)+A123
      XL(N2)=XL(N2)+A123
      XL(N3)=XL(N3)+A123
  999 CONTINUE
      CC=EE/(X32*Y21-X21*Y32)/(1.-EENU*EENU)
      Z=.5*(1.-EENU)
      II=3*(MEM-1)
      IIP1=II+1
      IIP2=II+2
      IIP3=II+3
      STRS(IIP1,1)=Y32
      STRS(IIP1,2)=-EENU*X32
      STRS(IIP1,3)=-Y31
      STRS(IIP1,4)=EENU*X31
      STRS(IIP1,5)=Y21
      STRS(IIP1,6)=-EENU*X21
      STRS(IIP2,1)=EENU*Y32
      STRS(IIP2,2)=-X32
      STRS(IIP2,3)=-EENU*Y31
      STRS(IIP2,4)=X31
      STRS(IIP2,5)=EENU*Y21
      STRS(IIP2,6)=-X21
      STRS(IIP3,1)=-Z*X32
      STRS(IIP3,2)=Z*Y32
      STRS(IIP3,3)=Z*X31
      STRS(IIP3,4)=Z*Y31
      STRS(IIP3,5)=-Z*X21
      STRS(IIP3,6)=Z*Y21
      DO995K=IIP1,IIP3
      DO995J=1,6
  995 STRS(K,J)=STRS(K,J)*CC
  500 CONTINUE
 1000 CONTINUE
      DO1100I=1,N
 1100 XL(I)=XL(I)*RHO
 1200 CONTINUE
      GOTO3000
 2000 CONTINUE
```

```
      IDUM = 0
      DO2100I=1,M
      N1=NOD1(I)
      N2=NOD2(I)
      X2MX1=X(N2)-X(N1)
      Y2MY1=Y(N2)-Y(N1)
      EL=SQRT(X2MX1*X2MX1+Y2MY1*Y2MY1)
      CCC=X2MX1/EL
      SSS=Y2MY1/EL
      ECCL=EE*CCC*CCC/EL
      ESSL=EE*SSS*SSS/EL
      ESCL=EE*SSS*CCC/EL
      EKL(1)=ECCL
      EKL(2)=ESCL
      EKL(3)=-ECCL
      EKL(4)=-ESCL
      EKL(5)=ESSL
      EKL(6)=-ESCL
      EKL(7)=-ESSL
      EKL(8)=ECCL
      EKL(9)=ESCL
      EKL(10)=ESSL
  602 CONTINUE
      DO2095J=1,10
      IDUM=IDUM+1
 2095 AK(IDUM)=EKL(J)
      STRS(I,1)=-EE/EL*CCC
      STRS(I,2)=-EE/EL*SSS
      STRS(I,3)=-STRS(I,1)
      STRS(I,4)=-STRS(I,2)
  600 CONTINUE
      XL(I)=EL*RHO
 2100 CONTINUE
  601 CONTINUE
 3000 CONTINUE
      CALL FIXAK
      RETURN
      END


      SUBROUTINE FIXAK
COMMON CARDS:DATA,PRINT
      DO 100 I=1,NK
 100  IIB(I)=0
      DO 110 I=1,NB
      IF(IB(I).GT.1000) GO TO 120
      IDUM=IB(I)+IB(I)-1
      GO TO 110
 120  CONTINUE
      IDUM=2*(IB(I)-1000)
 110  IIB(IDUM)=1
      IF(ISITP.EQ.2)GOTO400
```

```
      DO 300 I=1,M
      N1=NOD1(I)+NOD1(I)-1
      N2=N1+1
      N3=NOD2(I)+NOD2(I)-1
      N4=N3+1
      N5=NOD3(I)+NOD3(I)-1
      N6=N5+1
      II=(I-1)*(21)
201   IF(IIB(N1).EQ.0) GO TO 202
      AK(II+1)=1.
      AK(II+2)=0.
      AK(II+3)=0.
      AK(II+4)=0.
      AK(II+5)=0.
      AK(II+6)=0.
202   IF(IIB(N2).EQ.0) GO TO 203
      AK(II+2)=0.
      AK(II+7)=1.
      AK(II+8)=0.
      AK(II+9)=0.
      AK(II+10)=0.
      AK(II+11)=0.
203   IF(IIB(N3).EQ.0) GO TO 204
      AK(II+3)=0.
      AK(II+8)=0.
      AK(II+12)=1.
      AK(II+13)=0.
      AK(II+14)=0.
      AK(II+15)=0.
204   IF(IIB(N4).EQ.0) GO TO 205
      AK(II+4)=0.
      AK(II+9)=0.
      AK(II+13)=0.
      AK(II+16)=1.
      AK(II+17)=0.
      AK(II+18)=0.
205   IF(IIB(N5).EQ.0) GO TO 206
      AK(II+5)=0.
      AK(II+10)=0.
      AK(II+14)=0.
      AK(II+17)=0.
      AK(II+19)=1.
      AK(II+20)=0.
206   IF(IIB(N6).EQ.0) GO TO 300
      AK(II+6)=0.
      AK(II+11)=0.
      AK(II+15)=0.
      AK(II+18)=0.
      AK(II+20)=0.
      AK(II+21)=1.
300   CONTINUE
      GOTO500
```

```
  400 CONTINUE
      DO450I=1,M
      N1=NOD1(I)+NOD1(I)-1
      N2=N1+1
      N3=NOD2(I)+NOD2(I)-1
      N4=N3+1
      II=(I-1)*10
  451 IF(IIB(N1).EQ.0)GOTO452
      AK(II+1)=1.
      AK(II+2)=0.
      AK(II+3)=0.
      AK(II+4)=0.
  452 IF(IIB(N2).EQ.0)GOTO453
      AK(II+2)=0.
      AK(II+5)=1.
      AK(II+6)=0.
      AK(II+7)=0.
  453 IF(IIB(N3).EQ.0)GOTO454
      AK(II+3)=0.
      AK(II+6)=0.
      AK(II+8)=1.
      AK(II+9)=0.
  454 IF(IIB(N4).EQ.0)GOTO450
      AK(II+4)=0.
      AK(II+7)=0.
      AK(II+9)=0.
      AK(II+10)=1.
  450 CONTINUE
  500 CONTINUE
  501 CONTINUE
      RETURN
      END


      SUBROUTINE MERGE
COMMON CARDS:DATA,PRINT,OPT,WORK
      DO 100 K=1,NT
 100  EK(K)=0.
      IF(ISITP.EQ.2)GOTO500
      IDUM=0
      DO 200 I=1,M
      N1=NOD1(I)
      N2=NOD2(I)
      N3=NOD3(I)
      TTT=(T(N1)+T(N2)+T(N3))/3.
      DO400J=1,21
      IDUM=IDUM+1
  400 EKL(J)=AK(IDUM)*TTT
      DO 40 JJ=1,21
      GO TO (7,1,2,1,3,1,4,2,1,3,1,5,1,3,1,4,3,1,6,1,4),JJ
 7    ND=2*(N1-1)+1
      L=ISUM(ND)
```

```
      GO TO 20
1     L=L+1
      GO TO 20
2     L=L+2*(N2-N1)-1
      GO TO 20
3     L=L+2*(N3-N2)-1
      GO TO 20
4     ND=ND+1
      L=ISUM(ND)
      GO TO 20
5     ND=2*(N2-1)+1
      L=ISUM(ND)
      GO TO 20
6     ND=2*(N3-1)+1
      L=ISUM(ND)
20    CONTINUE
40    EK(L)=EK(L)+EKL(JJ)
200   CONTINUE
      GOTO700
  500 CONTINUE
      IDUM=0
      DO600I=1,M
      N1=NOD1(I)
      N2=NOD2(I)
      DO610J=1,10
      IDUM=IDUM+1
  610 EKL(J)=AK(IDUM)*T(I)
      DO640JJ=1,10
      GOTO(607,601,602,601,604,602,601,605,601,604),JJ
  607 ND=2*(N1-1)+1
      L=ISUM(ND)
      GOTO620
  601 L=L+1
      GOTO620
  602 L=L+2*(N2-N1)-1
      GOTO620
  604 ND=ND+1
      L=ISUM(ND)
      GOTO620
  605 ND=2*N2-1
      L=ISUM(ND)
  620 CONTINUE
  640 EK(L)=EK(L)+EKL(JJ)
  600 CONTINUE
  700 CONTINUE
      RETURN
      END


      SUBROUTINE DCOP
COMMON CARDS:DATA,PRINT,WORK
      EK(1)=SQRT(EK(1))
```

```
      DUM=1./EK(1)
      JJ=NTIM(1)
      IF(JJ.EQ.1) GO TO 100
      DO 100 J=2,JJ
      EK(J)=EK(J)*DUM
100   CONTINUE
      KK=J
      DO 500 I=2,NK
      IM1=I-1
      JJ=I+NTIM(I)-1
      DO 500 J=I,JJ
      KK=KK+1
      SUM=0.
      DO  490 L=1,IM1
      ITST=NTIM(L)+L-1
      IF(J.GT.ITST) GO TO 490
      ID1=ISUM(L)+I-L
      ID2=ISUM(L)+J-L
      SUM=SUM+EK(ID1)*EK(ID2)
490   CONTINUE
      IF(I.NE.J) GO TO 495
      EK(KK)=SQRT(EK(KK)-SUM)
      DUM=1./EK(KK)
      GO TO 500
  495 EK(KK)=(EK(KK)-SUM)*DUM
 500  CONTINUE
      RETURN
      END


      SUBROUTINE CHOS
COMMON CARDS:DATA,PRINT,WORK
      DO950LC=1,NLC
  200 CONTINUE
      P(1,LC)=F(1,LC)/EK(1)
      DO 600 J=2,NK
      SUM=0.
      JM1=J-1
      DO  580 L=1,JM1
      ITST=NTIM(L)+L-1
      IF(J.GT.ITST) GO TO 580
      ID1=ISUM(L)+J-L
      SUM=SUM+EK(ID1)*P(L,LC)
 580  CONTINUE
      ID2=ISUM(J)
  600 P(J,LC)=(F(J,LC)-SUM)/EK(ID2)
      P(NK,LC)=P(NK,LC)/EK(NT)
      NKM1=NK-1
      DO 700 K=1,NKM1
      J=NK-K
      SUM=0.
      JJ=J+1
```

```
      ITST=NTIM(J)+J-1
      DO 680 L=JJ,NK
      IF(L.GT.ITST) GO TO 680
      ID1=ISUM(J)+L-J
      SUM=SUM+EK(ID1)*P(L,LC)
680   CONTINUE
      ID2=ISUM(J)
  700 P(J,LC)=(P(J,LC)-SUM )/EK(ID2)
803   CONTINUE
  950 CONTINUE
      RETURN
      END


      SUBROUTINE FIXU
COMMON CARDS:DATA,PRINT
      DO 500 K=1,NB
      ID4=IB(K)
      IF(ID4.GT.1000) GO TO 65
      IM=2*(ID4-1)+1
      GO TO 70
65    ID4=ID4-1000
      IM=2*ID4
70    CONTINUE
      DO700LC=1,NLC
  700 P(IM,LC)=0.
500   CONTINUE
      RETURN
      END


      SUBROUTINE GETS
COMMON CARDS:DATA,PRINT,WORK
      IF(ISITP.EQ.2)GOTO1000
  600 CONTINUE
      DO 500 I=1,M
      N1=NOD1(I)
      N2=NOD2(I)
      N3=NOD3(I)
      J=2*(N1-1)+1
      DO101LC=1,NLC
  101 U(1,LC)=P(J,LC)
      J=J+1
      DO102LC=1,NLC
  102 U(2,LC)=P(J,LC)
      J=2*(N2-1)+1
      DO103LC=1,NLC
  103 U(3,LC)=P(J,LC)
      J=J+1
      DO104LC=1,NLC
  104 U(4,LC)=P(J,LC)
      J=2*(N3-1)+1
```

```
      DO105LC=1,NLC
  105 U(5,LC)=P(J,LC)
      J=J+1
      DO106LC=1,NLC
  106 U(6,LC)=P(J,LC)
      II=3*(I-1)
      DO107LC=1,NLC
      DO108IS=1,3
      S(I,IS,LC)=0.
      DO108K=1,6
  108 S(I,IS,LC)=STRS(II+IS,K)*U(K,LC)+S(I,IS,LC)
  107 S(I,4,LC)=SQRT(S(I,1,LC)*S(I,1,LC)+S(I,2,LC)*S(I,2,LC)-
     XS(I,2,LC)*S(I,1,LC)+S(I,3,LC)*S(I,3,LC)*(3.))
  301 CONTINUE
  300 CONTINUE
 500  CONTINUE
      GOTO2000
 1000 CONTINUE
 1600 CONTINUE
      DO1500I=1,M
      N1=NOD1(I)
      N2=NOD2(I)
      J=2*N1-1
      DO1101LC=1,NLC
 1101 U(1,LC)=P(J,LC)
      J=J+1
      DO1102LC=1,NLC
 1102 U(2,LC)=P(J,LC)
      J=2*N2-1
      DO1103LC=1,NLC
 1103 U(3,LC)=P(J,LC)
      J=J+1
      DO1104LC=1,NLC
 1104 U(4,LC)=P(J,LC)
      DO1107LC=1,NLC
      S(I,1,LC)=0.
      DO1108K=1,4
 1108 S(I,1,LC)=STRS(I,K)*U(K,LC)+S(I,1,LC)
      S(I,2,LC)=0.
      S(I,3,LC)=0.
 1107 S(I,4,LC)=S(I,1,LC)
  200 CONTINUE
 1500 CONTINUE
 2000 CONTINUE
      RETURN
      END
```