GEOMETRIC PROGRAMMING:

METHODS FOR DEALING WITH DEGREES OF DIFFICULTY

BY

Lyle R. Wilson

A Thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science, Mathematics

Signed: ______________________

Golden, Colorado
Date: 9-9-1977

Approved: ______________________
Thesis Advisor

____
Head of Department

Golden, Colorado
Date: 9-9-1977

ABSTRACT

Before the development of geometric programming, non-linear optimization was a collection of specialized techniques -- many applicable for specific functions (for example, quadradic programming; where the objective function is a sum of quadratic terms). Problems not meeting these rigid format specifications were often unsolvable mathematically. Conversely, geometric programming can be applied to a larger set of non-linear optimization problems. Originally, application of geometric programming was limited to minimizing "posynomial" (positive polynomial) objective functions and constraints. Later, the technique was extended to include general polynomials as well as maximization problems.

Optimization by means of geometric programming more often than not, is a pencil and paper technique, where calculations are reduced to little more than solving a system of simultaneous linear equations. Little use, however, has been made of geometric programming when the degrees of difficulty is greater than zero because of the difficulty encountered in determining values of the dual variables.

This thesis presents techniques for extending the applicability of geometric programming to problems with degrees of difficulty greater than zero. Throughout the

thesis, numerical examples of non-linear optimization problems taken from technical journals and textbooks are included. The purpose of these problems, aside from illustrating how to optimize by means of geometric programming, is to show there is an alternative to calculus and Lagrangian functions. Because of the complexity and large number of computations necessary to optimize problems with degrees of difficulty greater than zero, digital computer programs have to be employed. Source listing of the programs used in this thesis are given and the programs are well documented.

CONTENTS

INTRODUCTION

In this study, some techniques for dealing with geometric programming problems having positive degree of difficulty are presented.[1] Generally, the use of geometric programming has been limited to some extent because of the additional computational effort necessary to solve those problems for which the degrees of difficulty exceed zero. The techniques presented in this thesis are easily used and readily adaptable to many problems whose degree of difficulty exceed zero. However, before these techniques are presented, a brief review of geometric programming is given, as well as the application of geometric programming to solve some practical engineering and design problems. These problems were selected from various technical journals and engineering textbooks for the purpose of illustrating that geometric programming can be a very useful problem solving technique for engineers and designers. Next, the notion of degrees of difficulty is presented, and then som methods of dealing with degrees of

---

[1]Degrees of difficulty is defined as the difference between the number of terms and the number of variables less one for a primal objective function.

difficulty are explored. Finally, the potential of geometric programming as a problem solving technique is assessed and areas for further research are suggested.

CHAPTER ONE - Review of Geometric Programming

1. Geometric Programming, Posynomials, and Unconstrained Optimization

The originators of geometric programming, Duffin, et al (1), defined a posynomial to be a polynomial whose coefficients are all positive, and the only restriction on the exponents of the independent variables is that each must be a real-valued constant. In mathematical notation, a posynomial can be written as:

$$(1.1.1) \quad G(\overline{X}) = C_1 x_1^{a_{11}} x_2^{a_{12}} \ldots x_n^{a_{1n}} + C_2 x_1^{a_{21}} x_2^{a_{22}} \ldots x_n^{a_{2n}} + \ldots + C_t x_1^{a_{t1}} x_2^{a_{t2}} \ldots x_n^{a_{tn}}$$

$$\sum_{i=1}^{t} C_i \prod_{j=1}^{n} x_j^{a_{ij}} , \qquad \text{where: } C_i \geq 0,\ i=1,2,\ldots t; \quad a_{ij} \equiv \text{real-valued constants} \quad x_j \geq 0,\ j=1,2,\ldots,n.$$

Many functions of the type (1.1.1) exist in engineering design and economic evaluation situations. An example is an overall operating cost equation, whose individual cost terms when summed describe the cost incurred for that particular operation. Note that in the above formulation a multitude of non-linear situations can be described, because the exponents are only restricted to be real-valued. Geometric programming was designed specifically to determine the minimum value of posynomials. For a more detailed development of geometric programming, the reader should refer to references (1) or (11).

Development of geometric programming was motivated by the fact that in many cases the minimum value of an unconstrained, non-linear objective function could be found rapidly and easily by exploiting the inequality relationship between the weighted arithmetic mean and the geometric mean of a set of positive numbers. Thus if $k_i, i = 1,2,3,\ldots, t$, is a set of positive numbers and $w_i, i = 1,2,3,\ldots, t$, is a set of non-negative weights, then the inequality relationship between the weighted arithmetic mean and the geometric mean is:

(1.1.2)
$$\sum_{i=1}^{t} w_i k_i \geq \prod_{i=1}^{t} k_i^{w_i}, \quad 0 \leq w_i \leq 1, \quad i=1,2,3,\ldots,t, \text{ and } k_i > 0.$$

When $k_1 = k_2 = \ldots = k_t$, the maximum value of the geometric mean will be exactly equal to the minimum value of the related arithmetic mean. By having an objective function in the form of a weighted arithmetic mean, Duffin, et al, (1, Chapter 1) showed how to develop the related geometric mean and how to determine its maximum value.

Briefly, the method is as follows:

(1.1.3) consider the inequality:

$$w_1k_1 + w_2k_2 + \dots w_tk_t \geq k_1^{w_1}k_2^{w_2}\dots k_t^{w_t}$$

where the k_i are non-negative numbers, and the w_i are arbitrary positive weights satisfying the condition of normality:

(1.1.4)

$$w_1 + w_2 + \dots + w_t = \sum_{i=1}^{t} w_i = 1.$$

with the change of variables:

(1.1.5)

$$Y_i = k_iw_i, \; i = 1,2,3,\dots t,$$

then the inequality (1.1.2) becomes:

(1.1.6) $$\sum_{i=1}^{t} Y_i \geq \prod_{i=1}^{t} \left(\frac{Y_i}{w_i}\right)^{w_i}.$$

The left hand side of (1.1.6) is known as the "primal function" or "primal program", and the right hand side is called the "predual function". Now, if we denote a posynomial as:

(1.1.7)
$$G = Y_1 + Y_2 + \dots + Y_t,$$

and G is a posynomial of the form (1.1.1), so that

(1.1.8)

$$Y_i = C_ix_1^{a_{i1}}x_2^{a_{i2}} \dots x_n^{a_{in}},$$

then the predual function has the form:

(1.1.9)
$$\left(\frac{C_1x_1^{a_{11}}x_2^{a_{12}}\cdots x_n^{a_{1n}}}{w_1}\right)^{w_1}\left(\frac{C_2x_1^{a_{21}}x_2^{a_{22}}\cdots x_n^{a_{2n}}}{w_2}\right)^{w_2}\cdots\left(\frac{C_tx_1^{a_{t1}}x_2^{a_{t2}}\cdots x_n^{a_{tn}}}{w_t}\right)^{w_t}\prod_{i=1}^{t}\left(\frac{C_i}{w_i}\right)^{w_i}\prod_{j=1}^{h}x_j^{B_j}$$

where the B_j are the linear combinations of the exponents, or:

(1.1.10)
$$B_j = \sum_{i=1}^{t} w_i a_{ij} \qquad , \; j= 1,2,3,\ldots,n.$$

It can be shown (see 1, p.7) that the maximum value of the predual function is achieved by selecting the weights so that the B_j vanish, and this maximum value of the predual function is equal to the minimum value of the primal function. A set of weights, w_i, $i= 1,2,3,\ldots,t$, that satisfy the condition of forcing the B_j to vanish leaves the predual as a function of the positive coefficients of the primal objective function and not the primal independent variables; that is:

(1.1.11)
$$\sum_{i=1}^{t}\left(\frac{C_i}{w_i}\right)^{w_i}\prod_{j=1}^{n}x_j^{B_j} = \prod_{i=1}^{t}\left(\frac{C_i}{w_i}\right)^{w_i} \qquad \text{if}$$

(1.1.12)
$$\sum_{j=1}^{n} B_j = \sum_{j=1}^{n} w_i^{a_{ij}} = 0, \; i = 1,2,3,\ldots, t.$$

when the B_j are forced to vanish the predual function, which is then not dependent upon primal variables, is termed the "dual function".

Thus, geometric programming offers some very attractive features that can make solving certain non-linear functions feasible. A feature not often found using traditional calculus-based algarithms is that geometric programming optimization can be accomplished by solving a system of simultaneous linear equations, namely the orthogonality conditions $\left(\sum_{j=1}^{n} B_j = 0\right)$, one for each variable, and the normality condition $\left(\sum_{i=1}^{t} w_i = 1\right)$. Another feature is that once the optimal values of the weights are known and substituted into the dual function, the optimal value of the primal function is immediately known. Optimal values of the <u>weights</u> remain unchanged even if the values of the <u>coefficients</u> change. Therefore, each weight relates the relative contribution that each primal term makes to the optimal value of the primal function.

As previously mentioned, using the optimal values of the weights in the dual function yields the minimum value of the primal objective function; however, the values of the primal variables yielding this optimal value are not known. These values can be obtained by remembering that the geometric programming concept was built upon the use of the weighted arithmetic mean. Therefore, at optimality each primal objective function term weighted by the value of its weight, contributes equally to the optimal solution; or:

$$\text{(1.1.13)} \quad \frac{c_i \prod_{j=1}^{n} x_j^{a_{ij}}}{w_i} = G_{min}(\bar{x}), \quad i = 1,2,3,\ldots t.$$

To better illustrate the principles outlined in this section, the next section illustrates the use of geometric programming to solve a chemical engineering problem.

2. Application of Geometric Programming to a Posynomial

The preceding method is illustrated here by applying it to determine the minimum value of a fluid dynamics problem (optimum economic pipe diameter), a problem discussed and solved by Peters and Timmerhour (6, Chapter 9) using the traditional method of differential calculus.

Stated simply, the problem is to determine the minimum cost for a piping system. The total annual cost for the piping and pumping system is approximated by:

(1.2.1)
$$\$(D_i) = \frac{0.273 q_f^{2.84} \rho^{0.84} \mu^{0.16} K(1+J)H_y}{D_i^{4.84} E} + (1\ F)XD_i^n K_f .$$

Definitions of the variables and the constants in the equation are:

<u>Constants</u>

g_f = fluid flow rate, cubic feet per second

ρ = fluid density, pounds per cubic feet

μ_c = fluid viscosity, centipoises

K = cost of electrical power, dollars per kilowatt hour

H_y = hours of yearly operation

E = efficiency of the motor and pumping, expressed as a fraction

J = frictional loss due to fittings and bends, expressed as equivalent fractional loss in straight pipe

F = ratio of total cost for fittings and installation to purchase cost for new pipe

X = purchase cost of new pipe per foot of pipe length if pipe is 1 inch in diameter, dollars per linear foot

K_f = fixed charges, including maintenance, expressed as a fraction of initial cost for completely installed pipe

n = a constant with value dependent on type of pipe

Variable

D_i = inside pipe diameter in inches

This problem has two cost terms; the first one being the pumping costs, the other term describes the fixed costs. Although the problem is very simple and using calculus would not prove too difficult, geometric programming reveals valuable information about the problem that calculus would not reveal.

For the purpose of simplifying the equation, the following representation of constants is made:

(1.2.2)
$$C_1 = \frac{0.273 q_f^{2.84} \rho^{0.84} \mu^{0.16} K (1+J) H_y}{E}$$

and:

(1.2.3)
$$C_2 = (1+F)\, X K_F$$

Substituting the above relations into (1.2.1), the problem can be restated as:

(1.2.4) Minimize total annual costs: $\$(D_i)=C_1D_1^{-4.84} + C_2D_1^n.$

It is easily seen that the above problem is a posynomial and the solution can be obtained by geometric programming.

The dual function for the above problem is stated as:

(1.2.5) Maximize:

$$D(\overline{w}) = \left(\frac{C_1}{w_1}\right)^{w_1}\left(\frac{C_2}{w_2}\right)^{w_2}$$

Subject to the constraints:

(1.2.6) $w_1 + w_2 = 1$ (normality condition)

(1.2.7) $-4.84w_1 + nw_2 = 0$ (orthogonality condition; one variable, one equation)

Solving the set of linear constraints (1.2.6), and (1.2.7) for the weights (w_1 and w_2) gives:

(1.2.8)
$$w_1 = \frac{n}{n+4.84},$$

(1.2.9)
$$w_2 = \frac{4.84}{n+4.84}.$$

After the substitution of the optimal values for the dual variables w_1 and w_2 into equation (1.2.5), the minimum total annual cost is immediately obtained as:

(1.2.10)

$$\underset{\min}{\$(D_i)} = \underset{\max}{D(\overline{w})} = \left(\frac{C_1}{\frac{n}{n+4.84}}\right)^{\frac{n}{n+4.84}} \left(\frac{C_2}{\frac{4.84}{n+4.84}}\right)^{\frac{4.84}{n+4.84}}$$

or after substituting the expressions for C_1 and C_2 yields:

(1.2.11)

$$\$(D_i) = \left(\frac{(n+8.84)\,0.273\,q_f^{2.84}\,\rho^{0.84}\,\mu^{0.16}K(1+J)H_y}{n}\right)^{\frac{n}{n\ 4.84}} \left(\frac{(1+F)XK_f(n+4.84)}{4.84}\right)^{\frac{4.84}{n+4.84}}$$

Suppose for example, that steel pipe is to be used for the piping system ($n = 1.5$), then the values of w_1 and w_2 are:

(1.2.12)

$$w_1 = \frac{1.5}{6.34} = 0.237$$

and:

(1.2.13)

$$w_2 = \frac{4.84}{6.34} = 0.763$$

These values indicate that the pumping costs contribute 23.7% of the total annual cost, while the fixed costs contribute 76.3%. This ratio is necessary for optimality, no matter what the cost coefficients in each term are. This ratio is never determined by the method of differential calculus. Further, the engineer can put this information to good use.

In the case that costs have to be reduced, he immediately knows the proportions necessary to reduce the other costs and still have an optimal design.

Knowing the optimal value of the objective function, and the optimal values of the weights, the optimal policy or optimum pipe diameter can be each weighted term contributes equally to the optimal answer. Thus, with the problem at hand:

(1.2.14)
$$\left(\frac{0.237q_f^{2.84}\rho^{0.84}\mu_c^{0.16}{}_cK(1+J)H_y(n+4.84)}{n}\right)^{\frac{n}{n+4.84}}$$

$$\left((1+F)XK_F(n+4.84)\right)^{\frac{4.84}{n+4.84}} = \left(\frac{0.237q_f^{2.84}\rho^{0.84}\mu_c^{0.16}K(1+J)}{D_i^{4.84}En}\right)$$

$$Hy(n+4.84) = \frac{(1+F)D_i^{n}K_F(n+4.84)}{4.84}$$

Using the first and third quantities of the above relationship and raising both to the $\left(\frac{n+4.84}{4.84}\right)$th power, relates:

(1.2.15)
$$\left(0.237q_f^{2.84}\rho^{0.84}\mu_c^{0.16}K(1+J)H_y(n+4.84)\right)^{\frac{n}{4.84}} = D_i^{\frac{n(n\ 4.84)}{4.84}}$$

After raising both sides of (1.2.14) to the $\left(\frac{4.84}{n}\right)$th power yields:

(1.2.16)
$$\left(\frac{0.237q_f^{2.84}\rho^{0.84}\mu_c^{0.16}K(1\ J)H_y(n\ 4.84)}{n}\right) = D_i^{n+4.84}$$

and finally:

(1.2.17)

$$D_{i_{opt}} = \left(\frac{0.237q_f^{2.84}\,\rho^{0.84}\,\mu^{0.16}K(1+J)H_y(n+4.84)}{n}\right)^{\frac{1}{n+4.84}}$$

To show that geometric programming yielded the optimal answer, differential calculus will be applied to the problem. The normal sequence is to differentiate the function with respect to each independent variable, and solve for the values of the variables which cause the derivatives to vanish, then determine if the conditions for minimization are met. Finally, substitute the values of the independent variables into the original cost function and obtain the optimal value of the cost function.

Applying the above technique to the problem at hand (using the same representation of constants as equations (1.2.2) and (1.2.3)), the first and second derivatives with respect to D_i are:

(1.2.18)

$$\frac{\partial \$}{\partial D_i} = -4.84C_1D_i^{-5.84}+nC_2D_i^{n-1}$$

and

(1.2.19)

$$\frac{\partial^2 \$}{\partial D_i^2} = (-5.84)\ (4.84)C_1D_i^{-6.84}\ \ n(n-1)C_2D_i^{n-2}.$$

Setting equation (1.2.18) equal to zero, solving for D_i and simplifying gives the equation for the optimal for D_i as:

$$(1.2.20)\qquad D_i = \left(\frac{4.84C_1}{nC_2}\right)^{\frac{1}{n+4.84}} = \left(\frac{0.273q_f^{2.84}\,\rho^{0.84}\,\mu^{0.16}K(1+J)H_y(n+4.84)}{n}\right)^{\frac{1}{n+4.84}}$$

the same as (1.2.16).

Upon substitution of the value of D_i, into (1.2.19), it is apparent that the conditions for optimality are met; that is $\frac{\partial^2\$}{\partial D_i^2} > 0$. The algebraic substitution of (1.2.20) into the cost function (1.2.11) yields, after some simplification, the same relation as (1.2.17). But, notice that with calculus no sensitivity of the result is readily available, and only after the optimal policy (value of the independent variable, D_i) had been determined and substituted back into the cost equation, was the optimal cost discovered. As stated before, geometric programming yielded the same answer as obtained by using calculus; however, looking back on the problem, geometric programming yielded the optimal solution plus additional information.

In concluding this section, some simple, easy-to-use rules that can be implemented in applying geometric programming to solve unconstrained posynomial optimization problems will be formulated.

Rule 1: Form the dual objective function from the given problem (1.1.1).

Maximize:

$$D(\overline{w}) = \left(\frac{\text{coefficient of 1st cost term}}{w_1}\right)^{w_1} \times \left(\frac{\text{coefficient of 2nd cost term}}{w_2}\right)^{w_2} \cdots \left(\frac{\text{coefficient of last cost term}}{w_{last}}\right)^{w_{last}}$$

Rule 2: Set up the normality and orthogonality conditions for the weights; solve the resultant set of linear equations for the optimal values of the weights.

(1) the normality constraint: the sum of the weights for the terms of the primal objective function must equal unity.

$$w_1\begin{pmatrix}\text{1st cost}\\ \text{term}\end{pmatrix} + w_2\begin{pmatrix}\text{2nd cost}\\ \text{term}\end{pmatrix} + \cdots + w_{last}\begin{pmatrix}\text{last cost}\\ \text{term}\end{pmatrix} = 1$$

(2) The orthogonality constraint: for each variable X_j, the exponent of the j^{th} variable in the i^{th} cost term times the weight for that cost term i, summed over all the cost terms, must equal zero.

For variable X_j:

$$\begin{pmatrix}\text{exponent of variable}\\ X_j\\ \text{in 1st cost term}\end{pmatrix} w_1 + \begin{pmatrix}\text{exponent of variable}\\ X_j\\ \text{in 2nd cost term}\end{pmatrix} w_2 + \cdots + \begin{pmatrix}\text{exponent of variable}\\ X_j\\ \text{in last cost term}\end{pmatrix} w_{last} = 0.$$

(3) Solve the resultant set of simultaneous linear equations for the optimal set of weights.

<u>Rule 3</u>: Using the optimal set of weights, solve for the maximum value of D ($\overline{w}$), hence the minimum value of the primal objective function.

<u>Rule 4</u>: Knowing the optimal value of the primal objective function and the optimal set of weights, solve for the values of the independent variables of the primal objective function. Accomplish this by using the relationship that at optimality, each primal cost term weighted by its optimal weight contributes equally to the optimal solution:

$$\frac{\text{1st cost term}}{w_{1,\,opt}} = \frac{\text{2nd cost term}}{w_{2,\,opt}} = \cdots = \frac{\text{last cost term}}{w_{\text{last},\,opt}}$$

$$= \underset{\text{Max.}}{D(\overline{w})} = \underset{\text{Min.}}{G(\overline{X})}.$$

3. Geometric Programming and Constrained Minimization of Posynomials

The preceding sections have dealt with optimization of problems in which the posynomial objective function (1.1.1) was not constrained. This section presents the results of Duffin, et al (1, p.78) in solving the most general posynomial optimization problems by geometric programming techniques.

In mathematical notation, the constrained posynomial optimization problem is:

(1.3.1) Minimize: $G_0(\bar{x}) = c_{01}x_1^{a_{011}}x_2^{a_{012}}\dots x_n^{a_{01n}} + c_{02}x_1^{a_{021}}x_2^{a_{022}}\dots x_n^{a_{02n}} + \dots + c_{0t_0}x_1^{a_{0t_01}}x_2^{a_{0t_02}}\dots x_n^{a_{0t_0n}}$

$$= \sum_{i=1}^{t_0} c_{0i} \prod_{j=1}^{n} x_j^{a_{0ij}}$$

(1.3.2) subject to:

$G_k(x) = c_{k1}x_1^{a_{k11}}x_2^{a_{k12}}\dots x_n^{a_{k1n}} + c_{k2}x_1^{a_{k1n}}\dots x_n^{a_{k2n}} + \dots + c_{kt_k}x_1^{a_{kt_k1}}x_2^{a_{kt_k2}}\dots x_n^{a_{kt_kn}} \leq 1$

$$= \sum_{i=1}^{t_k} c_{ki} \prod_{j=1}^{n} x_j^{a_{kij}}$$

for $k=1,2,\dots n$; and $x_j > 0$, for $j=1,2,3,\dots,n$.

Solution of the general posynomial optimization problems, equations (1.3.1) and (1.3.2), by means of geometric programming involves solving a dual problem similar in form to the dual problem for unconstrained optimization.

The dual function can be stated as:

(1.3.3) Maximize:

$$D(\overline{w}) = \prod_{i=0}^{m} \prod_{j=1}^{t_i} \left(\frac{c_{ij}w_{i0}}{w_{ij}}\right)^{w_{ij}}$$

where: $w_{00} \equiv 1$

subject to:

(1.3.4)

$$w_{01} + w_{02} + \cdots + w_{0t_0} = \sum_{i=1}^{t_0} w_{0i}$$

(normality condition for the primal objective function),

(1.3.5)

$$\sum_{k=1}^{m} \sum_{i=1}^{t_k} a_{kij}w_{ki} = 0, \; j=1,2,3,\ldots,n.$$

(orthogonality condition; one equation for each primal variable),

(1.3.6)

$$\sum_{i=1}^{m} w_{ij} \equiv w_{i0} \geq 0, \; j=1,2,3,\ldots,t_i.$$

(linear equality condition, one equation for each primal constraint, or positivity condition for the constraints),

as well as the non-negativity conditions for the weights:

(1.3.7) $w_{ij} \geq 0$, $i=1,2,3,\ldots,t_k$; $k=1,2,3,\ldots m$;

and $j=1,2,3,\ldots,n$.

Although the above notation is complex, the solution of the dual problem for a constrained primal problem is obtained in the same manner as for an unconstrained primal problem. Again, the solution procedure is to obtain the optimal values of the weights $(\overline{w})$ from the set of simultaneous linear equations (1.3.4) to (1.3.6), and then substitute these values into the dual objective function (1.3.3). Hence, the maximum value of the dual problem then corresponds to the minimum value of the primal problem.

Once the optimal values of the weights $(\overline{w})$ and the optimal value of the dual objective function are known, the values of the primal variables $(\overline{X})$ are found from the relationships:

(1.3.8)

$$\frac{C_{0i} \prod_{j=1}^{n} x_j^{a_{0ij}}}{w_{0i}} = G_{0\,min}(\overline{x}) = D_{max}(\overline{w}) \quad i = 1,2,3,\ldots,t_0,$$

and:

(1.3.9)

$$C_{ki} \prod_{j=1}^{n} x_j^{a_{kij}} = \frac{w_{ki}}{w_{k0}}$$

for $k = 1,2,3,\ldots,m$; and $i = m$; $i = 1,2,3,\ldots,t_k$,

Since there are at least as many cost terms as there are variables $(\overline{x})$, n equations can be found from which the optimal values of the variables $(\overline{x})$ can be determined. Solving for the values of $(\overline{x})$ can be simplified by performing the logrithmic transform of equations (1.3.8) or (1.3.9). Logrithmic transformation of these equations makes the relationships linear; thus:

$$(1.3.10)\quad \log\left(C_{0i}\prod_{j=1}^{n} x_j^{a_{0ij}}\right) = \log C_{0i} + \log x_1^{a_{0i1}} + \log x_2^{a_{0i2}}$$
$$+\dots+\log x_n^{a_{0in}} - \log w_{0i} = \log C_{0i} + a_{0i1}\log x_1 + a_{0i2}\log x_2$$
$$+\dots+a_{0in}\log x_n - \log w_{0i} = \log G_{0_{min}}(\overline{x})$$

for $i = 1,2,3,\dots,t_0$,

and:

$$(1.3.11)\quad \log\left(C_{kj}\prod_{j=1}^{n} x_j^{a_{kij}}\right) = \log C_{kj} + \log x_j^{a_{ki1}} + \log x_2^{a_{ki2}}$$
$$+\dots+\log x_n = \log C_{kj} + a_{ki1}\log x_1 + a_{ki2}\log x_2$$
$$+\dots+a_{kin}\log x_n = \log w_{ki} - \log w_{k0}$$

for $k = 1,2,3,\dots m$, and $i = 1,2,3,\dots,t_k$.

Classical solution techniques for non-linear constrained optimization problems involve the introduction of Lagrange multipliers. For each primal constraint, there is the corresponding product of a Lagrange multiplier and the constraint introduced into the primal objective function. With this procedure, the objective function is transformed into what is termed the "Lagrangian function" and produces an unconstrained optimization problem. Applying the Kuhn-Tucker conditions[2] to this problem, the method proceeds

---

[2]For an excellent discussion of the Kuhn-Tucker conditions see: Hillier, F.S., Liebermann, G.J., Introduction to Operations Research, Holden-Day, Inc., San Francisco,1967, pp. 575-578.

to differentiating the Lagrangian function with respect to each independent variable and each Lagrange multiplier. Values for the independent variables as well as the Lagrange multiplier are obtained which force the derivatives to vanish. The optimal value of the objective function is obtained by solving the Lagrangian function with the optimal set of independent variables and Lagrange multipliers.

Let's now take a cursory glance at how geometric programming would approach the optimization of a constrained problem by applying the method presented in this section to a simple problem.

4. Application of Geometric Programming to a Constrained Posynomial Optimization Problem

The problem presented in this section has already been utilized elsewhere (10) to illustrate the principles of geometric programming. It is being used again here because the algebraic manipulations won't overshadow the presentation of the principles.

Simply stated, the problem is to determine whether or not an oil tank can be rebuilt for less the $1,000. The cost for materials to reconstruct the tank is $\$1/ft^2$. As is the usual case in engineering design problems, the new tank must meet specific physical requirements. In this case, the tank must be constructed such that it will hold at least 1,000 gals of oil. Thus, the objective function has two terms; first the cost for the circular top, and second, the cost for the cylindrical body. Mathematically then, the primal objective function is:

(1.4.1) Minimize total costs

$$\$(R,H) = \underbrace{\pi R^2 Ft^2 x \frac{\$1}{ft^2}}_{\text{(cost for top)}} + \underbrace{2\pi RH ft^2 x \$1/ft^2}_{\text{(cost for body)}}$$

To constrain the above objective function the capacity constraint is:

(1.4.2)

$$\pi R^2 H \leq 1000$$

However, to solve the problem using geometric programming, the constraint must be expressed in the form of (1.3.2) or:

(1.4.3)
$$\frac{1000}{R^2H} \leq 1$$

This simple problem is therefore a function of two variables, R the radius in feet, and the height H in feet.

The geometric programming dual objective function is:

(1.4.4) Maximize:
$$D(\overline{w}) = \left(\frac{\pi}{w_{01}}\right)^{w_{01}} \left(\frac{2\pi}{w_{02}}\right)^{w_{02}} \left(\frac{1000}{w_{11}} w_{10}\right)^{w_{11}}$$

subject to the following constraints:

(1.4.5) $w_{01} + w_{02} = 1$ (normality condition)

(1.4.6) $2 w_{01} + w_{02} - 2 w_{11} = 0$ (orthogonality condition for R)

(1.4.7) $w_{02} - w_{11} = 0$ (orthogonality condition for H)

(1.4.8) $w_{11} = w_{10} \geq 0$ (linear inequality constraint)

Simplification of equation (1.4.4), $w_{11} = w_{10}$, leads to:

(1.4.8) Maximize:
$$D(\overline{w}) = \left(\frac{\pi}{w_{01}}\right)^{w_{01}} \left(\frac{2\pi}{w_{02}}\right) \left(1000\right)^{w_{11}}$$

Solution for optimal values of the weights proceeds smoothly and therefore we obtain:

(1.4.9) $w_{02} = w_{11} = 2/3$

(1.4.11) $w_{01} = 1/3$

Substitution of these values of the weights into equation (1.4.4) yields the minimum value of \$(R,H):

(1.4.12) $\$_{min}(R,H) = D(\overline{w}) = \left(\frac{\pi}{1/3}\right)^{1/3} \left(\frac{2\pi}{2/3}\right)^{2/3} \left(1000\right)^{2/3} =$

$\left(3\pi\right)^{1/3} \left(3\pi\right)^{2/3} \left(1000\right)^{2/3} = 300\pi \doteq 942.$

Therefore, the optimal solution indicates that an optimal design can be built for less than \$1,000. However, in the course of solving the problem we have obtained additional information about the relative costs. It has been determined that the cost for the top is one-half that incurred for the body ($w_{01} : w_{02} = 1/3 : 2/3 = 1/2$). This ratio is invariant and must be present or the design is sub-optimal. What is needed now is to solve for the optimal policy that yields the minimum cost of \$$300\pi$.

Values of the independent variables, R and H, are readily obtained by utilizing the fact that an optimality, each cost term weighted by its appropriate weight contributes equally to the optimal solution, and the relationship for constraints (1.3.9). In this case:

(1.4.13) $$\frac{\pi R^2}{1/3} = \frac{2\pi RH}{2/3} = 300\pi$$

and

(1.4.14) $$\frac{1000}{R^2H} = \frac{w_{11}}{w_{10}} = 1$$

Using the first and third quantities of (1.4.13), then

(1.4.15) $R^2 = 100$, or $R = 10$;

next using either the second and third quantities of (1.4.13) or (1.4.15) yields:

(1.4.16) $R = H = 10$.

Admittedly the problem of this section is very simple; hence, the reader can verify by differential calculus that geometric programming did indeed yield the optimal solution. The remainder of this section will be devoted to outlining some simple rules to apply for optimizing constrained posynomial functions.

Some simple, easy-to-use rules for applying geometric programming to constrained posynomial optimization can be stated as follows:

Rule 1: Formulate the dual objective function from the primal problem.

$$\text{Maximize: } D(\overline{w}) = \left(\frac{\text{coefficient of 1st cost term}}{w_{01}}\right)^{w_{01}} \left(\frac{\text{coefficient of 1st cost term}}{w_{02}}\right)^{w_{02}}$$

$$\cdots \left(\frac{\text{coefficient of last cost term}}{w_{0,last}}\right)^{w_{0,last}} \left(\frac{\text{coefficient of 1st term 1st constraint}}{w_{11}}\right)^{w_{11}}$$

$$\left(\frac{\text{coefficient of 2nd term 1st constraint}}{w_{12}}\right)^{w_{12}} \cdots \left(\frac{\text{coefficient of last term 1st constraint}}{w_{1last}}\right)^{w_{1last}}$$

$$\cdots \left(\frac{\text{coefficient of 1st term last constraint}}{w_{last,1}}\right)^{w_{last,1}} \left(\frac{\text{coefficient of last term, 1st constraint}}{w_{last,2}}\right)^{w_{last,2}}$$

$$\cdots \left(\frac{\text{coefficient of last term last constraint}}{w_{last,last}}\right)^{w_{last,last}} .$$

Rule 2: Form the normality, orthogonality, and linear inequality conditions for the weights and solve the resultant set of linear equations for the optimal values of the weights.

(1) The normality condition: The sume of the weights for the terms of the primal objective function must equal unity.

$$\underset{\left(\begin{array}{c}\text{1st cost}\\ \text{term}\end{array}\right)}{w_{01}} + \underset{\left(\begin{array}{c}\text{2nd cost}\\ \text{term}\end{array}\right)}{w_{02}} + \cdots + \underset{\left(\begin{array}{c}\text{last cost}\\ \text{term}\end{array}\right)}{w_{0,last}} = 1.$$

(2) The orthogonality condition: For each variable j, the exponent of the j^{th} variable in equation k, term i multiplied times the weight for that term, summed over all the terms must equal zero.

for variable j, in equation k:

$$\underset{\left(\begin{array}{l}\text{exponent of } x_{jin}\\ \text{kth equation,}\\ \text{1st term}\end{array}\right)}{a_{k1j} \times w_{k1}} + \underset{\left(\begin{array}{l}\text{exponent of } x_{jin}\\ \text{kth equation}\\ \text{2nd term}\end{array}\right)}{a_{k2j} \times w_{k2}} + \cdots + \underset{\left(\begin{array}{l}\text{exponent of } x_{jin}\\ \text{kth equation}\\ \text{last term}\end{array}\right)}{a_{k,last,j} \times w_{k,last}} = 0$$

(3) The positivity conditions for the constraints (one equation for each constraint). The sum of the weights associated with a given constraint must equal the weight associated with that constraint, which in turn must be greater than zero.

For example, the k^{th} constraint:

$$w_{k1} + w_{k2} + \cdots + w_{k,last} = w_{k0} \geq 0.$$

Rule 3: Solve the simultaneous set of linear equations from Rule 2 for the optimal values of the weights.

Rule 4: Solve for the optimal value of the primal function using the dual objective function from Rule 1 and the optimal values of the weights from Rule 3.

Rule 5: Determine the optimal policy (optimal values of the primal variables) by using the fact that at optimality, each term of the primal objective function weighted by its individual weight contributes equally to the optimal solution or:

$$\frac{\text{1st cost term}}{w_{o1}} = \frac{\text{2nd cost term}}{w_{o2}} = \cdots = \frac{\text{last cost term}}{w_{o,\,last}}$$

$$= G_{o\,\min}(\overline{x}) = D_{\min}(\overline{w}).$$

Alternatively taking advantage of the positivity condition for the constraints, each term of a constraint times its optimal weight is equal to the portion that the constraint weights the dual function.

Thus, for constrain k,

$$w_{k1} \times \begin{matrix}1^{st} \text{ term of} \\ k^{th} \text{ constraint}\end{matrix} = w_{k2} \times \begin{matrix}2^{nd} \text{ term of} \\ k^{th} \text{ constraint}\end{matrix} \cdots$$

$$= w_{ko}.$$

5. General Polynomials and Geometric Programming

Geometric programming was generalized beyond the previous discussions to allow negative coefficients in the primal objective function as well as the constraints, and also to permit reversed inequality constraints. The following is a brief summary of the most general formulation of a geometric programming problem. For a more complete discussion of these extensions and results, refer to (9) and (5).

A mathematical definition of the general polynomial function[3] which is to be minimized is:

(1.5.1) Minimize $$H(\overline{x}) = \sum_{i=1}^{t_0} \sigma_{0i} c_{0i} \prod_{j=1}^{n} x_j^{a_{0ij}}$$

subject to the polynomial constraints,

(1.5.2) $$H_k(\overline{x}) = \sum_{i=1}^{t_k} \sigma_{ki} c_{ki} \prod_{j=1}^{n} x_j^{a_{kij}} \leq \sigma_k \quad \equiv \pm 1 \qquad \sigma_{ki} \equiv \pm 1$$

and

(1.5.3) $x_j > 0,\ j = 1,2,3,\ldots n.$

Corresponding to the above primal problem, the geometric programming dual problem is:

(1.5.4) Maximize:
$$D(\overline{w}) = \sigma \left[\prod_{k=1}^{m} \prod_{i=1}^{t_k} \left(\frac{c_{ki} w_{k0}}{w_{ki}} \right)^{\sigma_{ki} w_{ki}} \right]^{\sigma}$$

---

[3]A general polynomial is defined as a polynomial whose coefficients and exponents are restricted to being real valued.

subject to the following linear constraints:

(1.5.5) (the normality condition)

$$\sum^{t_0} \sigma_{0i} w_{0i} = \sigma \quad (\equiv \pm 1)$$

(1.5.6) (the orthogonality conditions, for each of the primal independent variables.

$$\sum_{k=1}^{m} \sum_{i=1} \sigma_{ki} a_{kij} w_{ki} \leq 1$$

and

(1.5.7) $w_{ki} > 0$, $k = 0,1,2,3,\ldots,m$; and $i = 1,2,3,\ldots,t_k$.

In the preceding notation, minimization of posynomials (equation 1.1.1) is a special case of the above formulation (i.e., $\sigma = 1$), so this presentation is for the most general optimization technique.

Again, the reader will notice that even the most nonlinear optimization problems can be reduced to solving a set of simulaneous linear equations for optimal values of the weights. Once the values of the weights have been determined, these values are substituted into the dual objective function (1.5.4), and the minimum value of the primal problem can be determined. After the maximum value of (1.5.4) is known, it is simply a matter of algebra to determine the values of the primal variables.

In the following section, the use of the preceding discussion will be used to solve an example problem.

It is hoped that the example will illustrate the application of the above equations which at first glance might appear quite formidable.

6. Optimization of a Non-Posynomial

The problem presented in this section is rather an elementary one; however, the application of the preceding section can be easily followed by the reader. A problem discussed and solved by Johnson (4, p.336) is typical of many optimization problems facing an engineer who is called upon not to minimize a cost function but maximize a design function. In this case, the design is for a torsion bar spring. It is desired to maximize the potential energy storing capacity of the spring system such that the spring is not bent or fatigued beyond some point where it might fail for a maximum impressed force.

Thus, the problem is to maximize the total potential energy of the spring system at any instant of time or:

(1.6.1) Maximize: $PE_{total} = PE_t + PE_b$, where

(1.6.2) PE_t = potential energy of twisting $= \dfrac{32(1+\mu)F^2R^2L}{E\pi d^4}$

and

(1.6.3) PE_b = potential energy of bending $= \dfrac{32F^2b^2L}{3\pi E}$

The constraint on the problem, relating the anticipated maximum impressed force, the safety factors, and the spring material strength is:

$$(1.6.4) \quad \frac{(1.8)(16.0)FR}{d^3} \leq \frac{S_e}{(1+P)N} .$$

Definitions of the variables and constants in the preceding problem are:

Constants

μ = Poisson's ratio

E = modulus of elasticity

F = anticipated maximum impressed force to be applied to the spring

b = distance between the moment arm and the bearing support for the torsion bar

L = total length of the torsion bar

Se = published fatigue strength factor for the material from which the bar is constructed

P = the ratio Se/St, where St is the yield strength for the bar

N = the safety factor, see the discussion in (4, Chapter 16)

Variables

d = diameter of the torsion bar

R = length of the moment arm

After combining the constants, a restatement of the problem is simply:

(1.6.5) Maximize: $PE_{total}(R,d) = K_1R^2d^{-4} + K_2d^{-4}$

(1.6.6) Subject to: $K_3Rd^{-3} \leq 3$

where,

(1.6.7) $$K_1 = \frac{32(1+\mu)F^2L}{\pi E},$$

(1.6.8) $$K_2 = \frac{32F^2b^2L}{3\pi E},$$

and

(1.6.9) $$K_3 = \frac{(1.8)(16.0)F(1+P)N}{S_eN}.$$

Notice that to conform the problem to the notation of the previous section we must minimize the negative of (1.6.5); therefore, $\sigma = -1$.

The geometric programming dual objective function is therefore,

(1.6.10) Maximize:

$$D(\overline{w}) = -1\left[\left(\frac{K_1}{w_{01}}\right)^{-w_{01}}\left(\frac{K_2}{w_{02}}\right)^{-w_{02}}\left(\frac{K_3w_{10}}{w_{11}}\right)^{w_{11}}\right]^{-1}$$

(1.6.11) Subject to: $(-1)w_{01} + (-1)w_{02} = -1$

(1.6.12) $(-1)w_{01} + w_{11} = 0$

(1.6.13) $(-1)2w_{01} + (-1)4w_{02} + 3w_{11} = 0$

(1.6.14) $$w_{11} = w_{10} \rightarrow 0$$

Solution of the system of equations (1.6.11) to (1.6.14 is:

(1.6.15) $$w_{01} = 2/3$$

(1.6.16) $$w_{02} = 1/3$$

(1.6.17) $$w_{11} = w_{10} = 4/3$$

Substitution of the above values into (1.6.10) yields:

(1.6.18) $$-\underset{\min}{\text{PEtotal}}(R,d) = \underset{\max}{D}(\overline{w}) = -\left(\frac{27K_1^2\, K_2}{4K_3^4}\right)^{1/3}$$

Utilizing equation (1.5.8) relates that,

(1.6.19) $$\frac{-K_1R^2d^{-4}}{2/3} = \frac{-K_2d^{-4}}{1/3} = -\underset{\max}{D}(\overline{w})$$

Solving the equations (1.6.15) and (1.6.16) for the optimal values of R and d (the primal variables) gives:

(1.6.20) $$R = \left(\frac{2K_2}{K_1}\right)^{1/2} = \left(\frac{2b^2}{3(1+\mu)}\right)^{1/2}$$

and

(1.6.21) $$d = \left(\frac{2K_2K_3^2}{K_1}\right)^{1/6} = \left(\frac{32b^2F^2(1+\mu)n^2}{S_e\,\pi}\right)^{1/6}$$

The solution (1.6.20) and (1.6.21), as well as the relation for the optimal value of potential energy (1.6.18), corresponds exactly with that given by Johnson (4, p.374). Before concluding this section a caution should be made concerning the optimal values of the weights when one has a polynomial in which the σ_{0i} multipliers are not all positive (nonposynomial objective functions). The caution is that these weights are no longer constrained to lie in the interval $0 < w_{0i} \leq 1$. Therefore, the weights do not have the same connotation as was given in preceding sections. An example would be the following set of optimal weights, whose sum is indeed unity:

$$(1.6.22) \quad (1)\ 2/3 + (-1)\ 4/3 + (1)\ 5/3 = 1.$$

However, it is difficult to assign 4/3 or 5/3 contribution of a term to a total cost.

7. Degrees of Difficulty

The notion of "degrees of difficulty"[4] was coined by the innovators of geometric programming (1) as a qualitative measure of the effort necessary to solve for the optimal weights ($\overline{w}$) which maximize the dual function $D(\overline{w})$. Up to this point in the presentation, the practical problems that have been optimized by geometric programming were selected from a special class of problems. The problems were chosen such that the number of terms minus the number of independent variables less one equals zero. That is to say, the degree of difficulty is zero for these problems, and they were readily optimized by solving the set of simultaneous linear equations (the constraint set for the dual program) for the optimal values of the weights.

If the degree of difficulty for a geometric programming dual program is greater than zero, additional computational effort is necessary to determine the solution for which the dual function $D(\overline{w})$ is maximized. The extra effort needed to solve problems with degrees of difficulty greater than zero is that there are more dual variables than equations (1.5.5 and 1.5.6) to determine the values of

[4]Degree of difficulty is given by T-N-1, where T = number of primal terms and N = number of primal variables.

the dual variables. Thus, for example, if the degree of difficulty equals one, the dual objective function D(w) must be maximized over the range of feasible values for one weight. The other weights can be expressed as functions of this one weight by solving the constraint set of the dual problem for the functional relationships.

A method of solving geometric programming problems with one degree of difficulty or more is to "linearize" the dual function by log transformation. Next, substitute into the log-transformed dual program the functional relationships of the weights. Then by differentiating this log-transformed dual objective function with respect to the weights (the number of differentials equals the degree of difficulty), and determining the values of the weights which force the derivatives to vanish. An example of this procedure follows.

Suppose we wish to minimize the function,

(1.7.1) Minimize: $4X_1^{-1}X_2^{-1} + 4X_2 + 2X_1 + 5X_1X_2$

The geometric programming dual function is,

(1.7.2) Maximize: $D(\overline{w}) = \left(\frac{4}{w_1}\right)^{w_1} \left(\frac{4}{w_2}\right)^{w_2} \left(\frac{2}{w_3}\right)^{w_3} \left(\frac{5}{w_4}\right)^{w_4}$

Log transforming (1.7.2) yields:

(1.7.3) $\text{Log } D(\overline{w}) = w_1 \log\left(\frac{4}{w_2}\right) + w_2 \log\left(\frac{4}{w_2}\right) + w_3 \log\left(\frac{2}{w_3}\right) \; w_4 \log\left(\frac{5}{w_4}\right)$

The normality and orthogonality equations are[5]:

(1.7.4) $w_1 = w_1$

(1.7.5) $w_2 = 1 - 2w_1$

(1.7.6) $w_3 = 1 - 2w_1$

(1.7.7) $w_4 = 3w_1 - 1$

Substitution of equations (1.7.4) through (1.7.10) into (1.7.3) gives,

(1.7.8) $$\operatorname{Log} D(w) = w_1 \log \frac{4}{w_1} \quad (1-2w_1)\log \frac{4}{1-2w_1}$$
$$(1-2w_1)\log \frac{2}{1-2w_1} \quad (3w_1-1)\log \frac{5}{3w_1-1} \quad .$$

Differentiation of (1.7.8), setting equal to zero, and reexponentiating leads to the expression:

(1.7.9) $$\left(\frac{4}{w_1}\right)\left(\frac{5}{3w_1-1}\right)^3 - \left(\frac{4}{1-2w_1}\right)^2\left(\frac{2}{1-2w_1}\right)^2 = 0.$$

Simplification of (1.7.9) is the polynomial equation,

(1.7.10) $$1568w_1^4 - 3568w_1^3 + 2586w_1^2 - 984w_1 + 125 = 0.$$

By iteratively approximating the solution for w_1, the solution for this problem is:

(1.7.11) $w_1 = 0.4158$

---

[5] A note of caution is needed here. w_1 cannot be allowed to take on values that will make w_2 or w_3 negative or zero, otherwise the positivity conditions on dual variables will be violated.

Using the solution (1.7.11) for W_1 and substituting these values into the relations (1.7.4) to (1.7.7) gives as optimal values for the weights

(1.7.12) $w_2 = 0.1684$

(1.7.13) $w_3 = 0.1684$

(1.7.14) $w_4 = 0.2474$

The above optimal values of the weights substituted into (1.7.2) gives the optimal value for D(w) of 15.45. Optimal values for the primal variables are:

(1.7.15) $x_1 = 0.906$

$x_2 = 0.453$

From the foregoing procedure, it might appear that even a simple problem can be made difficult. However, the traditional calculus approach of optimizing the primal problem is non-trivial. Note with geometric programming optimization was accomplished with one variable, while optimization of the primal problem involves two independent variables. It should be realized, of course, that as the degree of difficulty increases, additional effort is required by the above procedure to achieve the optimal solution by geometric programming. A matrix technique for those problems with degree of difficulty greater than 1 is very tedious computationally (the method employs the Newton-Raphson method). Therefore, a computer program which performs the calculations would be desirable. There are some disadvantages to using such programs. In particular, coding such a program could be a monumental task, and further, such matrix operations are inherently slow and expensive. The second chapter of this thesis presents some alternative methods of attack for problems with degrees of difficulty greater than zero.

CHAPTER TWO - Techniques to Employ in Dealing with Degrees of Difficulty

1. A Technique for One Degree of Difficulty

Often it occurs that by examining the constraints of the dual problem, one can obtain information about the weights that will allow one to determine an approximate optimum. This technique takes into account that if the primal problem is a posynomial, the weights for the dual objective function are bounded such that,

(2.1.1) $0 < w_i \leq 1.$

But, suppose it were possible to obtain more restrictive bounds on one weight for a one degree of difficulty problem. This would allow one to bound the optimal answer. If one desires to obtain a more accurate estimate, a graphical plot could be utilized to select new bounds, make another plot (thereby obtaining better resolution), etc., until the desired accuracy was obtained. Thus, no messy differentiation or solving complex polynomials would be needed.

To illustrate this technique, the problem of determining the economic order quanity (Q) will be used. This particular problem is discussed in (8) and an iterative technique for obtaining a solution is also detailed in this article. Essentially, this economic model is stated as, (2.1.2) Minimize: $K(Q)=20Q^{-1}+\frac{1}{48,000}Q+\frac{1}{576}\left(1+\frac{Q}{200}+\frac{Q}{120,000}\right)$

or more simply,

(2.1.3) Minimize:

$$K(Q)-\frac{1}{576}=K(Q)=20Q^{-1}+2.951\text{x}10^{-5}\ Q+1.447\text{x}10^{-8}+Q^{2}.$$

The geometric programming dual problem is,

(2.1.4) Maximize:

$$D(\overline{w})=\left(\frac{20}{w_1}\right)^{w_1}\left(\frac{2.951\text{x}10^{-5}}{w_2}\right)^{w_2}\left(\frac{1.447\text{x}10^{-8}}{w_3}\right)^{w_3}$$

(2.1.5) Subject to: $w_1+w_2+w_3=1$

(2.1.6) $-w_1+w_2+2w_3=0$

Note that the primal problem has three terms and only one variable (Q); therefore, its degree of difficulty is one.

By examining the constraint set for the dual objective function, we can solve for w_2,

(2.1.7) $w_2=\frac{1-3w_3}{2}$

However, w is bounded between 0 and +1, and applying these bounds on expression (2.1.7) produces the relation:

(2.1.8) $0<w_2=\frac{1-3w_3}{2}\leq 1,$

or after some simplification,

(2.1.9) $0<w_2\leq 1/3.$

Similarly, we can solve for w_3,

(2.1.10) $w_3=2w_1-1.$

However, by utilizing the relation (2.1.9) we can further bound expression (2.1.10), or

(2.1.11) $0 < w_3 = 2w_1 - 1 \leq 1/3$,

and after simplification determine that W_1 is bounded

(2.1.12) $1/2 \leq w_1 \leq 2/3$.

Bounds can be determined for w_2 by the same means; the bounds for w_2 are

(2.1.13) $0 < w_2 \leq 1/2$.

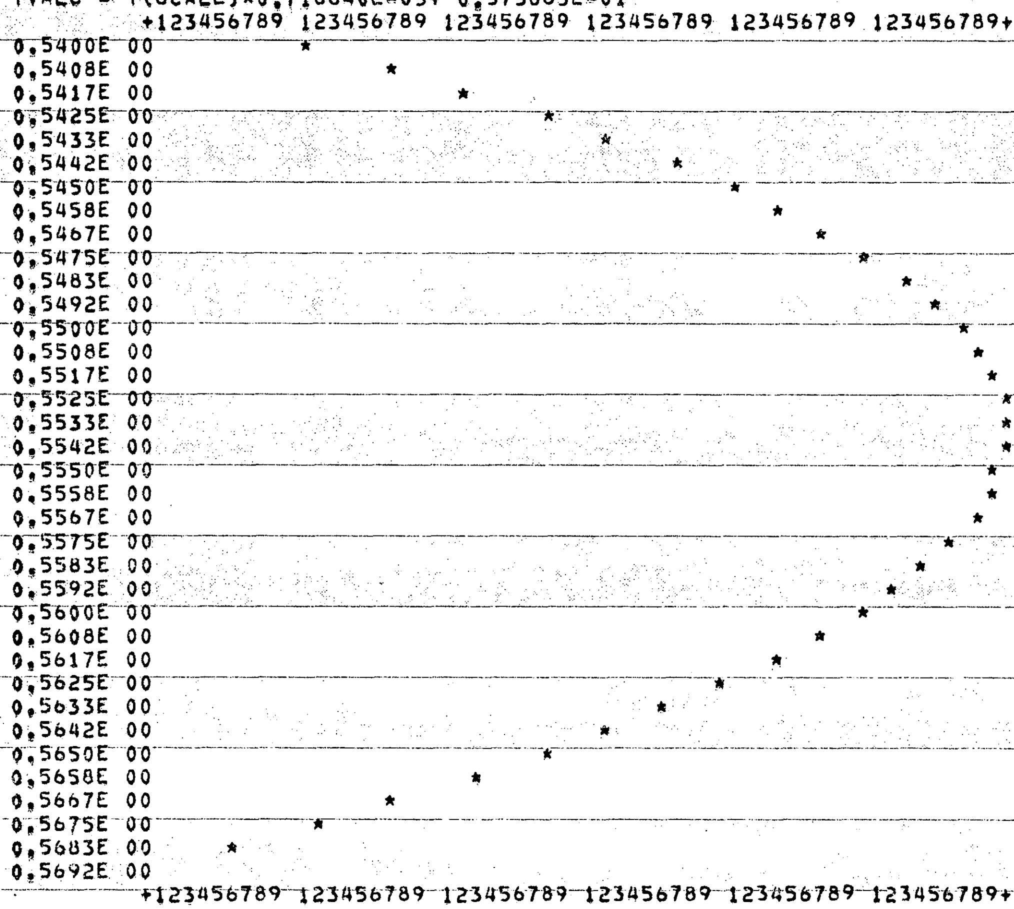
It appears then that the most restrictive bounds that can be ontained by this procedure are for w_1. If one were to solve the dual function (2.1.4) as a function of w_1 and evaluate that function at several values of w_1 within the bounds obtained and plot those values, a much more accurate value of the dual objective function could be obtained. However, that process could be very tedious. This thesis presents a FORTRAN computer program called LINE (see the Appendix) which evaluates the objective function and also makes a quick line-printer graph of the values. With such a program, the user can determine new limits and replot very quickly. It must be remembered that these graphs are limited with respect to accuracy because the program must scale the functional values to a whole number of horizontal character spaces.

The aforementioned graphical method was applied to the dual function (2.1.4). From the first output of the program, it was determined that the maximum value for the objective function occurred within the limits $0.54 \leq w_1 \leq 0.57$. Another pass was made with these new limits and the output for that run is shown in Figure 1. The plot of Figure 1 indicates that the optimal value of the objective function is 0.0578 and that $w_1 = 0.554$ is the optimal value for the weight w_1. The value of 0.0578 agrees with the answer given in the cited article.

A note about execution times for these graphs should be made. Output times for these graphs were approximately 45 seconds, including input/output operations using a DEC-10 computer system in a time-sharing environment.

This plotting procedure might raise the question of plotting the primal problem in a similar manner as the dual program. The question can be put aside because geometric programming duality theory (1, Chapter 4) assures that if the primal program is a posynomial, the dual problem is concave. Therefore, the graphical technique will determine an optimal value for the dual function. For those primal functions where there are negative coefficients or several

MAXIMUM Y FOUND = 0.577977E-01
MINIMUM Y FOUND = 0.573665E-01
YVALU = Y(SCALE)*0.718640E-05+ 0.573665E-01



inequality constraints (i.e., where some of the signum functions are negative), concavity is difficult to assess. Thus, the graphical technique may or may not determine the optimal value. However, it is likely that by graphical means one might determine whether or not concavity is present.

In this section a graphical technique to deal with one degree of difficulty problems has been presented. The next section presents a method to attack those problems which have more than one degree of difficulty.

2. Problems With Greater Than One Degree of Difficulty

For problems with greater than one degree of difficulty the initial research indicated that possibly the cartographic technique of contouring would be a good method of attack for at least those problems with two degrees of difficulty. Briefly, a contour map depicts a three-demensional surface -- one having length, breadth, and varying height above reference plane or datum -- on a two-dimensional piece of paper. On such a map, lines called contours are drawn to portray the intersections of the function surface with a series of horizontal planes at definite intervals above (or below) the datum plane. This method proved to be workable, but highly inefficient. In general, the technique investigated was to construct a contour map of the function, where the contours represented equal values of the dual objective function; visually select a smaller region in which the dual function obtained a maximum; replot the map; continue until resolution was sufficient. However, to produce a map 10 inches by 10 inches required at least 6,000 function evaluations. Also, selection of the datum plane with an appropriate contour interval could not always be specified or calculated with any certainity so that an acceptable map could be obtained. Thus to obtain a reasonable map of the function, at least two passes with the program were necessary to determine the best contouring parameters. It was then decided to include the two degree of difficulty problems with the higher dimensional problems and utilize another optimum seeking method.

The method finally selected was direct search. Various direct search techniques were tried (Hooke-Jeeves pattern search, golden section search, Rosen's gradient projection method), but the parallel tangents method proved the best for the problems tested. The technique of parallel tangents for dual problems with degrees of difficulty greater than one generally required less than 200 objective function evaluations to determine the optimum value of the dual function accurately. In fact, a two-degree of difficulty test problem only required 28 function evaluations. Details of the parallel tangents technique will not be give here, but rather the reader is referred to (3), and the appendix. The method was coded by this author in FORTRAN, and the code as well as the documentation for the program are given in the appendix. Advantages of parallel tangents over other direct search techniques are: complex matrix operations are avoided, excessive computer core storage is not required, and accurate results can be quickly obtained with the minimum number of function evaluations.

Optimization of dual functions by parallel tangents using program PARTAN proceeds first by reducing the number of weights which the algorithm must perturb or adjust during the search. This is accomplished by selecting a set of weights (the number of weights selected equaling the degrees of difficulty); solving the remaining weights as functions of the selected weights. Next, substitute the functions of the weights; and finally execute program PARTAN.

To illustrate the application of parallel tangents in optimizing a problem with greater than one degree of difficulty, the following function discussed by Smith (7) will be optimized by using geometric programming. Smith solved this problem by using the Lagrange multiplier technique and an iterative trial and error solution. This problem is concerned with determining the optimum production lot size which minimizes total production inventory cost. Smith's generalized production inventory problem is:

(2.2.1) Minimize: $$\sum_{i=1}^{N} C_i = \sum_{i=1}^{N} (C_{ci} q_i/2 \; C_{ri} d_i/q_i)$$

(2.2.2) Subject to: $$\sum_{i=1}^{N} \frac{d_i h_i}{q_i} \leq H$$ (set up time for the bottleneck machine),

where H = total annual setup hours available on the bottleneck machine

Constants

h_i = setup hours required on the bottleneck machine to produce one lot of product i

C_i = sum of the annual replenishment and inventory of product i

C_{ci} = annual cost of carrying one unit of product i in inventory

C_{ri} = replenishment cost incurred each time a lot of product i is produced

d_i = annual demand rate for product i

Variables

q_i = lot size in physical units for product i

Smith used the following values

i	d_i	C_{ci}	C_{ri}	h_i
1	200	10	250	10
2	400	40	180	40
3	600	20	240	100

and H = 500. Thus, the problem (with N = 3) becomes

(2.2.3) Minimize: $C_t(q)=5q_1 + 50{,}000q_1^{-1} + 20q_2 + 72{,}000q_2^{-1} + 10q_3 + 144{,}000q_3^{-1}$

(2.2.4) Subject to:
$$4q_1^{-1} + 32q_2^{-1} + 120q_3^{-1} \leq 1.$$

The corresponding geometric programming problem is

(2.2.5) Maximize: $D(\overline{w}) = \left(\frac{5}{w_{01}}\right)^{w_{01}}\left(\frac{5x10^4}{w_{02}}\right)^{w_{02}}\left(\frac{20}{w_{03}}\right)^{w_{03}}$
$$\left(\frac{72x10^4}{w_{04}}\right)^{w_{04}}\left(\frac{10}{w_{05}}\right)^{w_{05}}\left(\frac{1.44x10^5}{w_{06}}\right)^{w_{06}}\left(\frac{4w_{10}}{w_{11}}\right)^{w_{11}}$$
$$\left(\frac{32w_{10}}{w_{12}}\right)\left(\frac{120w_{10}}{w_{13}}\right)^{w_{13}}$$

(2.2.6) Subject to:

$$
\begin{array}{lr}
w_{01} + w_{02} + w_{03} + w_{04} + w_{05} + w_{06} & = 1 \\
w_{01} - w_{02} \qquad\qquad\qquad\qquad - w_{11} & = 0 \\
w_{03} - w_{04} \qquad\qquad\qquad - w_{12} & = 0 \\
w_{05} - w_{06} \qquad\qquad - w_{13} & = 0 \\
w_{11} + w_{12} + w_{13} & = w_{10} > 0.
\end{array}
$$

Notice that the problem has (9 terms - 3 variables - 1) five degrees of difficulty. With five degrees of difficulty, all the weights can be expressed as terms of five weights; and these expressions can be substituted into (2.2.5) to yield

(2.2.7) Maximize:
$$\left(\frac{5}{w_{01}}\right)^{w_{01}}\left(\frac{5x10^4}{w_{02}}\right)^{w_{02}}\left(\frac{20}{w_{03}}\right)^{w_{03}}\left(\frac{7.2x10^4}{w_{04}}\right)^{w_{04}}$$

$$\left(\frac{10}{1-w_{01}-w_{02}-w_{03}-w_{04}-w_{06}}\right)^{\substack{1-w_{01}-w_{02}-w_{03}\\-w_{04}-w_{06}}}\left(\frac{1.44x10^{-5}}{w_{06}}\right)^{w_{06}}$$

$$\left(\frac{4(1-2w_{02}-2w_{04}-2w_{06})}{w_{01}-w_{02}}\right)^{\substack{w_{01}\\-w_{02}}}\left(\frac{120(1-2w_{02}-2w_{04}-2w_{06})}{1-w_{01}-w_{02}-w_{03}-w_{04}-2w_{06}}\right)^{\substack{1-w_{01}-w_{02}\\-w_{03}-w_{04}-2w_{06}}}.$$

Bounds for the selected weights in this example are

(2.2.8) $w_{01} \geq w_{02}$

(2.2.9) $w_{03} \geq w_{04}$

(2.2.10) $0 < w_{01} \leq \frac{1}{2}$

(2.2.11) $0 < w_{02} \leq \frac{1}{2}$

(2.2.12) $0 < w_{03} \leq \frac{1}{2}$

(2.2.13) $0 < w_{04} \leq \frac{1}{2}$

(2.2.14) $0 < w_{06} \leq \frac{1}{2}$.

Execution of program PARTAN with the above relations determined that the optimal value was 6299.75 and identified the optimal solution for the selected weights as

(2.2.15) $w_{01} = 0.0866$

(2.2.16) $w_{02} = 0.0737$

(2.2.17) $w_{03} = 0.2699$

(2.2.18) $w_{04} = 0.1343$

(2.2.19) $w_{06} = 0.1114.$

Substitution of the above values into the relations for the other weights yields

(2.2.20) $w_{05} = 0.3251$

(2.2.21) $w_{11} = 0.0129$

(2.2.22) $w_{12} = 0.1356$

(2.2.23) $w_{13} = 0.2137$

(2.2.24) $w_{10} = 0.3622.$

By utilizing the relationship that at optimality each cost term weighted by its appropriate weight contributes equally to the optimal production inventory cost, the optimal quantities are

(2.2.25) $q_1 = 108.73$

(2.2.26) $q_2 = 85.13$

(2.2.27) $q_3 = 204.33,$

which agree with Smith's results.

It might be suggested that optimization of generalized polynomials by geometric programming and parallel tangents might involve unnecessary computation and effort as opposed to calculus or some other optimization technique. However, this technique doesn't involve evaluation of simultaneous complex differential equations or Lagrangian functions.

After some experience with program PARTAN, this author has determined that the most important parameters that the user can supply to the successful execution of the program are the boundary conditions. Although the dual function might be defined outside the domain of the weights, any function value obtained, of course, is meaningless. Also, the boundary conditions can be set up so as to keep PARTAN executing and not generating diagnostic messages or systems errors. This is accomplished by setting a variable equal to its boundary value when the boundary value is violated. Another fact which seems to help the execution is to start the search algorithm at the midpoint of the boundaries of each of the variables.

CHAPTER THREE

Conclusions

Chapter two illustrates optimization of polynomials can be accomplished when one is faced with degrees of difficulty. These techniques extend the usefulness of geometric programming, and with these extensions geometric programming becomes a good technique for optimization. Geometric programming lends itself to a wide variety of types of problems, and for many problems is the only realistic means of attack.

However, limitations of geometric programming still exist and prohibit one from rushing blindly into application. As mentioned before, the procedures of Chapter two will only succeed in those cases in which the dual function is concave. Concavity becomes troublesome for those problems in which one or more of the signum functions are negative. There seems to be no systematic way to ascertain whether or not a dual problem has the property of concavity.

As with all zero-gradient methods, the methods of Chapter two run the risk, in some cases, of locating a stationary point which is not a global optimum. When all the signum functions are positive, the dual function will be concave, and the solution determined will be a global optimum. With negative signums, however, the character of the dual function is uncertain and a local optimum may be found.

When faced with such a problem, one must vary the starting point for the algorithm and attempt to determine all the local optima.

Because the parallel tangents method is somewhat sensitive to starting points, selection of the starting point for the algorithm is more art than science, and experience has shown that the algorithm may not converge to any optimum at all, or in some cases may wander about for a few iterations before the optimum is reached. Of course, all of these factors are dependent upon the problem. When the number of dual variables for which the function must be optimized is large, the parallel tangents method does not posses round-off problems as do other direct search techniques. This fact makes the technique more attractive than algorithms which must invert or manipulate matrices.

In the previous chapter it was shown that some problems at first glance appear not to be solvable by geometric programming, and some variable transformation or substitution must be performed. Among the reasons that geometric programming cannot be applied directly is that the positivity conditions for primal constraints is violated or they contain functions other than polynomials or posynomials. An example of the latter is:

(3.1.1) Minimize: $Y(X) = K_1 e^{-K_2 X}\ K_2 X.$

Solving the above problem by calculus presents no problem. However, for geometric programming, even approximating the exponential form by the limiting form

$$(3.1.2) \quad \lim_{\epsilon \to 0} \left(1 + \frac{u(X)}{\epsilon} \right) = e^{u(X)}$$

does not help because $-K_2X$ is not a single-termed posynomial. Thus, the problem cannot be solved directly by geometric programming. With a change of variables though, one can obtain a solution by geometric programming. Upon obtaining the optimal solution, values of the original variables can be determined by the inverse transformation. For this case, let

$$(3.1.3) \quad K_2 X = \ln T$$

then,

$$(3.1.4) \quad e^{K_2 X} = T,$$

and

$$(3.1.5) \quad X = (1/K_2) \ln T = \ln T^{1/K_2}$$

Substitution of these relations into the original primal function we obtain

$$(3.1.6) \quad \text{Minimize: } Y'(T) = K_1 T^{-1} + (K_3/K_2) \ln T$$

but

$$(3.1.7) \quad \ln T = \epsilon^{-1} T - \epsilon^{-1},$$

therefore, our primal program has been transformed to

$$(3.1.8) \quad \text{Minimize: } X''(T) = Y'(T) + \epsilon^{-1}(K_3/K_2) = K_1 T^{-1} + \epsilon^{-1}(K_3/K_2))T \ .$$

The transformed problem is solvable by geometric programming and in fact the solution is (after letting $\epsilon \to 0$)

(3.1.9) $X = (k_1 K_2)/K_3$

Transformation procedures such as illustrated above have some shortcomings. If one is not clever enough to discover the change of variables which will transform the term into a polynomial form, the problem cannot be solved by geometric programming. The non-negativity of the primal constraints can also be avoided in a similar manner, however, this condition is not as obvious as the non-polynomial functions.

A suggestion to users of geometric programming that merits further investigation is that when formulating optimization programs, resist reducing the number of primal independent variables by assuming some of them constant. Degrees of difficulty might be avoided or be reduced by allowing more variables to enter into the primal program. This is the area to which engineers and designers should direct their efforts, because of their familiarity with the problems the equations describe.

In conclusion, it is this author's opinion that geometric programming has wide application to mathematical models with polynomial objective functions. Despite the limitations, geometric programming is a valuable mathematical tool for optimization of non-linear problems.

BIBLIOGRAPHY

1. Duffin, R.J., E.L. Peterson and C. Zener, Geometric Programming, John Wiley and Sons, 1967.

2. Duffin, R.J., "Dual Programs and Minimum Cost," SIAM Journal of Applied Math., Vol 10, No. 1, March, 1962.

3. Harkins, A., "The Use of Parallel Tangents in Optimization," Chem. Eng. Prog. Symposium Series, Vol. 59, No. 8, 1963.

4. Johnson, R.C., Optimum Design of Mechanical Elements, John Wiley and Sons, 1961.

5. Passy, U. and D.J. Wilde, "Generalized Polynomial Optimization," SIAM Journal of Applied Math., Vol. 15, No. 5, Sept., 1967.

6. Peters, M.S., and K.D. Timmerhaus, Plant Design and Economics For Chemical Engineers, McGraw-Hill, 1968.

7. Smith, S.B., "Economic Lot Sizes With a Restriction on Setup Hours," Journal American Production and Inventory Control Society, Vol. 11, No. 2, 2nd Qtr., 1969.

8. Thompoulos, N.T., and M. Lehman, "Effects of Inventory Obsolence and Uneven Demand on the EOQ Formula," Journal American Production and Inventory Control Society, Vol. 12, No. 4, 4th Qtr., 1971.

9. Wilde, D. J., and C.S. Beightler, Foundations of Optimization, Prentice-Hall, 1967.

10. Woolsey, R.E.D., G.A. Kochenburger, and K.R. Linck, "Can Engineers Find Happiness Without Calculus?," Hydrocarbon Processing, Aug., 1971.

11. Zener, C., "Cost Optimization Problems Treated by Geometric Means," Operations Research, Vol. 10, No. 1, 1962.

APPENDIX

I. PROGRAM LINE

Program LINE is a simple line printer plot program. The program consists of four parts:

(1) A main program which determines the x and y scale factors for the graph; controls the output for the function values; and labeling for both axes.

(2) A subroutine (FUNPTS) which determines the minimum and maximum values of the function for the range of x values.

(3) A set of utility routines

  (a) A routine which fills the print array with blanks (BLNX)

  (b) A routine which fills the print array with y axis information (BORDER)

  (c) A routine which prints the print array (PRINTM)

(4) A function subprogram (FUNVAL) which must be supplied by the user.

The steps which LINE performs to construct a graph are

(1) Determine the x increment

$$\text{x increment} = \frac{\text{maximum x value} - \text{minimum x value}}{\text{length of the graph times 6}} .$$

Since the x axis is the vertical axis, the program assumes there are six line printer characters per inch. If not, the program can be easily modified to accomodate other print character sizes.

(2) Evaluate the function at every x value, store these values in an array, as well as determine the maximum and minimum values. The x values are calculated once the x increment has been calculated from the relationship

x_i = x increment times i + minimum x.

(3) Determine the y scale factor from the equation

$$\text{y scale factor} = \frac{\text{maximum y - minimum y}}{\text{6 inches times 10 characters/inch}}.$$

(4) Blank the print array, fill the print array with the top border, and print the border along with the y scaling information.

(5) Calculate the number of horizontal spaces for each y value, where the number of spaces is calculated from the equation

$$\text{horizontal spaces} = \frac{y_i \text{ - minimum y}}{\text{y scale factor}}.$$

(6) Fill the print array with the number of horizontal spaces less one, and put an "*" in the last space. If the number of spaces is less than one, no "*" or blanks are needed.

(7) For every x value and y value print on the same line the x value followed by the print array.

(8) Go back to step 5 until all the x values have been printed.

(9) Fill the print array with the bottom border and then print the border.

Because the logic of the program is not complicated, no flow chart will be given and only the FORTRAN code with an example function is included. Only one line (or card) is needed for input to this program. The structure for this data is

Columns	FORTRAN Format Specification	Data Description
1 - 10	E10.5	XMAX the maximum x value
11 - 20	E10.5	XMIN the minimum x value
21 - 30	E10.5	SIZE the size of the graph vertically if this value is greater than 15 inches, the program resets this value equal to 15 inches

II. THE PARALLEL TANGENTS ALGORITHM

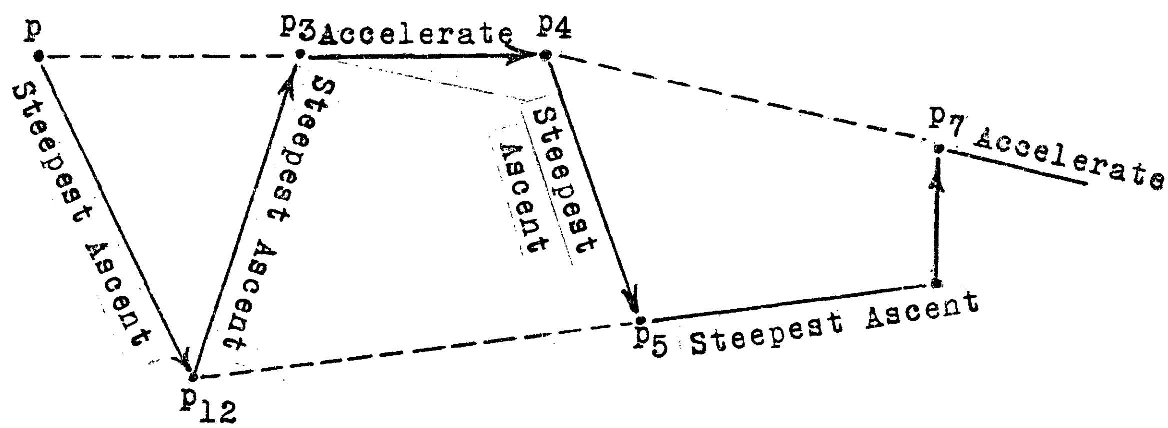
The parallel tangents algorithm may be summarized as follows (refer to the figure below).

(1) At the starting point p_0, evaluate the direction of steepest ascent.

(2) Proceed in the direction of steepest ascent until a high point along this vector is found, p_2.

(3) At p_2 evaluate the direction of steepest ascent and determine the high point p_3 of this vector.

(4) At point p_3 construct a new vector p_3-p_0; accelerate along this vector until the high point p_4 is reached.



. . .From point p_3 to termination the algorithm is. . .

(5) At p_i (i even) determine the direction of steepest ascent; proceed along this vector until the high point $p_{i\ 1}$ is found.

(6) At p_i (i odd) build the vector p_i-p_{i-3}, accelerate along this vector until its high point is determined.

(7) If an acceleration step at any point p_i fails to improve the objective function, restart the procedure at p_{i-1} (step 5).

Vector Searching

Determining the high point along a vector is accomplished by means of a golden section search. However, to employ a search, a base point on a vector instead of an interval is used for the "first experiment". This trial point is placed an arbitrary distance from the base point. If this trial point succeeds in improving the objective function, a new distance is determined by multiplying the trial distance by $\frac{1+\sqrt{5}}{2}$. This is repeated until a trial (experiment) fails to improve the objective function. When the failure occurs, the interval containing the high point is located between the (n-2)th experiment and the nth trial, with the (n-1)th experiment dividing the interval into a golden section. The vector search routine then uses the traditional golden section search algorithm to bracket the high point more accurately.

High points along vectors are not determined precisely because this would slow the convergence. Rather, the steepest ascent step is allowed to make progress toward the optimum. Specifying one to five experiments seems to give best performance.

Steepest Ascent

Directions of steepest ascent are determined as follows

(1) Establish the range for each variable, R_i .

(2) Evaluate $\Delta F_i = F(X_1, X_2, \ldots, X_i + \alpha R_i, X_{i\ 1}, \ldots X_n) - F(X)$

(3) Initial perturbations δ_i along the vector of steepest ascent are evaluated for each independent variable, where

$$\delta_i = \frac{\beta R_i \, \Delta F_i}{|\Delta F_i| \max} .$$

β is initially set to 0.1, thus the variable yielding the maximum ΔF_i will only be perturbed 1/10 of its range. If the first trial results is a success then β is multiplied by $1 + \frac{1+\sqrt{5}}{2}$. Multiplication is repeated until a failure occurs, then the parallel tangents algorithm proceeds to bracket the high point using the vector search technique. However, if the multiplication fails, then β is repeatedly divided by $1 + \frac{1+\sqrt{5}}{2}$ until a success occurs or β is reduced below some value ϵ . Values of α used in the steepest ascent portion should be 10^{-4} to 10^{-6}.

<u>Acceleration Step</u>

The first trial from point p_i (i odd) is set to 1/3 the distance $p_i - p_{i-3}$. If this trial yields an improvement in the objective function, the vector search proceeds in the manner described before. However, if the initial trial fails, two contractions of the step size are allowed; if after two contractions no successful trial is obtained, the parallel tangents algorithm is restarted at point p_{i-1}. During the acceleration step the interval between p_i and p_{i-3} is not searched.

Termination

Optimization is assumed to be completed when the step size β is reduced (by the vector search routine) to a value less than ϵ. Typical values of ϵ are 10^{-2} to 10^{-4}.

III. PROGRAM PARTAN

The program is coded in FORTRAN IV and consists of six distinct routines:

(1) The main program which contains the necessary logic for the parallel tangents algorithm.

(2) The vector search algorithm (Subroutine VECSCH)

(3) The steepest ascent algorithm (Subroutine STPASC)

(4) The output routine (Subroutine WRITER)

(5) The user supplied function subprogram (Function OBJFUN)

(6) The user supplied routine which assures that independent variables are not adjusted beyond their boundary values (Subroutine BOUNDS). Examples of user supplied routines are given with the source code listing of PARTAN. Both of the user routines must be supplied even if the boundary routine is only a dummy routine. The structure of these routines are given in the internal documentation of PARTAN.

INPUT DATA FOR PARTAN

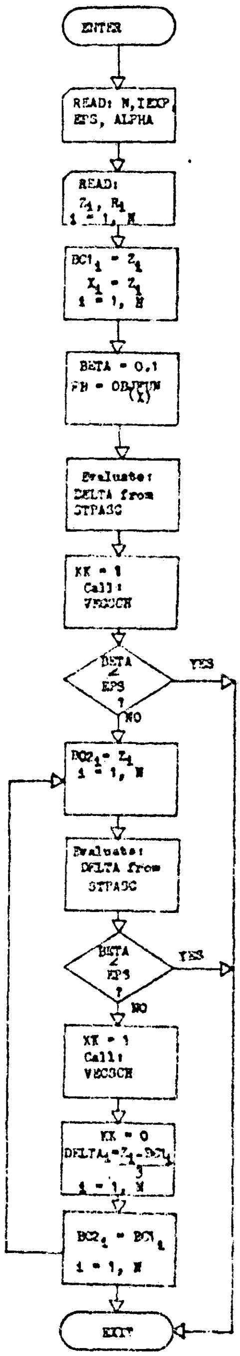
Card No.	Parameter Description	FORTRAN Format Specification	Punch Card Columns
1	N, number of independent variables	I5	1 - 5
1	IEXP, number of experiments to be used for bracketing high point of a vector	I5	6 - 10
1	EPS, termination criteria	E10.5	11 - 20
1	ALPHA, the fraction of the range that each variable is perturbed for determining steepest ascent	E10.5	21 - 30
2 to N+1	Z(I), starting value for the ith variable	E10.5	1 - 10
2 to N+1	R (I), range for the ith variable	E10.5	11 - 20.

FLOW CHARTS FOR PARTAN

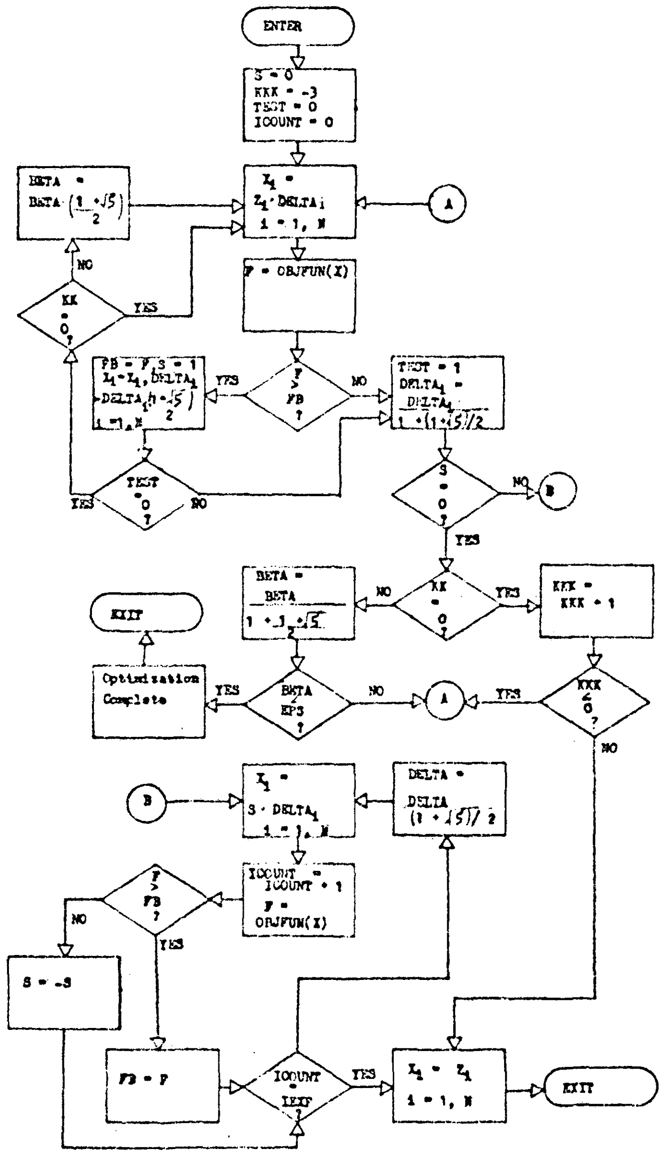
The following page details the logic of the parallel tangents algorithm incorporated into the FORTRAN coding.

LOGIC FLOW DIAGRAMS FOR THE PARALLEL TANGENTS ALGORITHM

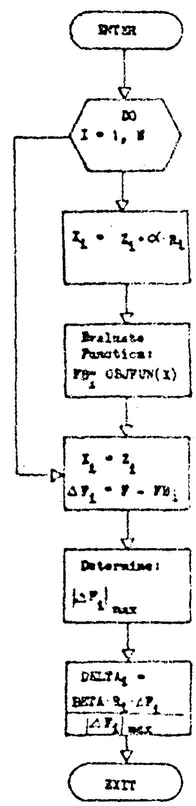
MAIN
PROGRAM



VECTOR SEARCH
SUBROUTINE
(VECSCH)



STEEPEST ASCENT
SUBROUTINE
(STPASC)



4. Source Listing for Program LINE

```
C      *****************************************************************
C      PROGRAM   LINE
C      LINE PRINTER PLOT OF Y = F(X)
C      MAXIMUM SIZE OF PLOT 6 INCHES HORIZONTAL (THE Y AXIS)
C      BY 15 INCHES VERTICAL (X AXIS)
C
C      XMAX = MAXIMUM X
C      XMIN = MINIMUM X
C      YMAX = MAXIMUM Y
C      YMIN = MINIMUM Y
C      Y IS AN ARRAY OF CALCULATED Y=F(X) VALUES
C      M IS THE PLOT ARRAY (ONE-DIMENSIONAL ARRAY)
C
      DIMENSION Y(80),M(70)
      DATA ISTAR/1H*/
C
C      READ IN THE MAX MIN X DESIRED, AND THE LENGTH OF THE PLOT VERTICALLY
C
      READ(5,1) XMAX,XMIN,SIZE
C
C      CHECK SIZE IF GREATER THAN 15 INCHES  SET SIZE = 15 INCHES.
C
      IF(SIZE.GT.15.) SIZE=15.
    1 FORMAT(3F10.0)
C
C      COMPUTE THE NUMBER OF ITERATIONS (ASSUMING LINE SPACING IS 6/INCH)
C
      K=SIZE*6.
C
C      COMPUTE THE X INCREMENT
C
      XINC=(XMAX-XMIN)/FLOAT(K)
C
C      FILL THE Y ARRAY AND OBTAIN YMAX, YMIN FOR SCALING PURPOSES
C
      CALL FUNPTS(K,XMIN,XINC,Y,YMAX,YMIN)
C
C      COMPUTE Y SCALE FACTOR  (YSCALE)
C
      YSCALE=(YMAX-YMIN)/60.
C
C      PRINT THE SCALING INFORMATION
C
      WRITE(6,2) YMAX,YMIN,YSCALE,YMIN
    2 FORMAT(1H ,18HMAXIMUM Y FOUND = ,E12.6,/,1H , 18HMINIMUM Y FOUND =
     1 ,E12.6,/,1H 17HYVALU = Y(SCALE)*,E12.6,2H+ ,E12.6)
C
C      BLANK THE PRINT ARRAY
C      FILL THE PRINT ARRAY WITH BORDER.
C      PRINT THE BORDER
C
      CALL BLNX(M,60)
      CALL BORDER(M,6)
      CALL PRINTM(M,70)
C
```

```
C       BLANK THE PRINT ARRAY
C       COMPUTE THE X
C       COMPUTE THE NUMBER OF BLANK SPACES FOR THE PRINT ARRAY
C       PUT A * AT Y=F(X) IN THE PRINT ARRAY
C       PRINT OUT THE PRINT ARRAY
C
      CALL BLNX(M,60)
      DO 200 I=1,K
      X=XMIN+(I-1)*XINC
      KK=AINT((Y(I)-YMIN)/YSCALE-.5)
      IF(KK.LT.1) GO  TO 100
      CALL BLNX(M,KK)
      M(KK+1)=ISTAR
      KL=KK+1
      WRITE(6,3) X,(M(J),J=1,KL)
      GO TO 200
  100 WRITE(6,4) X
  200 CONTINUE
    3 FORMAT(1H ,E10.4,80A1)
    4 FORMAT(1H ,E10.4)
C
C     FILL THE PRINT ARRAY WITH THE FINAL BORDER
C       PRINT THE FINAL BORDER
C
      CALL BORDER(M,6)
      CALL PRINTM(M,70)
      STOP
      END
C   *****************************************************************
C
C       SUBROUTINE TO BLANK OUT THE PRINT ARRAY FOR THE 1 TO JTH POSITIONS
C
      SUBROUTINE BLNX(N,J)
      DIMENSION N(1)
      DATA IBLNK/1H /
      DO 100 I=1,J
      N(I)=IBLNK
  100 CONTINUE
      RETURN
      END
C
C       SUBROUTINE TO PRINT THE PRINT ARRAY FOR THE 1 TO JTH POSITIONS
C
C   *****************************************************************
      SUBROUTINE PRINTM(M,J)
      DIMENSION M(1)
      WRITE(6,1) (M(I),I=1,J)
    1 FORMAT(1H ,80A1)
      RETURN
      END
C   *****************************************************************
C
C       SUBROUTINE TO FILL THE PRINT ARRAY WITH A BORDER FOR SCALING
C       THE 1 TO 9 POSITIONS ARE BLANK; THE 10TH POSITION IS 'ZERO'
C
      SUBROUTINE BORDER(M,J)
```

```
      DIMENSION M(1),IAR(9)
      DATA IAR/1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9/
      DATA IBLNK/1H /
      DATA IPLS/1H+/
      JPLS1=J+1
      DO 100 I=1,9
  100 M(I)=IBLNK
      M(10)=IPLS
      DO 200 I=2,JPLS1
      K=I*10
      M(K-9)=IAR(1)
      M(K-8)=IAR(2)
      M(K-7)=IAR(3)
      M(K-6)=IAR(4)
      M(K-5)=IAR(5)
      M(K-4)=IAR(6)
      M(K-3)=IAR(7)
      M(K-2)=IAR(8)
      M(K-1)=IAR(9)
  200 M(K)=IBLNK
      M(JPLS1*10)=IPLS
      RETURN
      END
C    *****************************************************************
C
C       SUBROUTINE TO FILL THE Y ARRAY AND FIND YMAX, YMIN
C
      SUBROUTINE FUNPTS(J,XMIN,XINC,Y,YMAX,YMIN)
      DIMENSION Y(1)
      YMAX=-1.E-10
      YMIN=1.E10
      DO 100 I=1,J
      X=XMIN+(I-1)*XINC
      Y(I)=FUNVAL(X)
      YMAX=AMAX1(YMAX,Y(I))
      YMIN=AMIN1(YMIN,Y(I))
  100 CONTINUE
C
C       USER MUST SUPPLY HIS OWN FUNCTION SUBPROGRAM CALLED 'FUNVAL(X)'
C
      RETURN
      END
C    *****************************************************************
C
C       THIS IS AN EXAMPLE FUNCTION SUBPROGRAM
C       THIS IS THE FORTRAN CODING OF THE EOQ MODEL
C
      FUNCTION FUNVAL(X)
      W01=X
      W02=2.-3.*W01
      W03=2.*W01-1.
      A=W01*ALOG10(20./W01)
      A=A+W02*ALOG10(2.951E-5/W02)
      A=A+W03*ALOG10(1.447E-8/W03)
      A=10.**A
      A=A+1./576.
      FUNVAL=A
      RETURN
      END
```

```
C      PROGRAM PARTAN--PARALLEL TANGENTS OPTIMIZING TECHNIQUE USING
C      VECTOR SEARCHING AND STEEPEST ASCENT ACCELERATING PROCEDURE.
C
C      NOTATION OF VARIABLES USED IN THIS PROGRAM AND SUBROUTINES
C
C      BC1,BC2     STORAGE ARRAYS FOR PAST VALUES OF Z, USED IN COMPUTATION
C                  OF ACCELERATION STEPS
C      F           VALUE OF OBJECTIVE FUNCTION AT ANY TRIAL X
C      FB          VALUE OF OBJECTIVE FUNCTION AT LAST SUCCESS FB=F(Z)
C      KK,KKK,I,S  VARIABLES USED IN SETTING PROGRAM LOGIC
C      TEST, J
C
C      N           NUMBER OF INDEPENDENT VARIABLES
C      R           VECTOR OF RANGES FOR THE INDEPENDENT VARIABLES
C      X           VECTOR INDEPENDENT VARIABLES TO BE USED FOR A GIVEN
C                  FUNCTION EVALUATION
C      Z           ARRAY OF INDEPENDENT VARIABLE VALUES THAT YIELDED LAST
C                  IMPROVEMENT OF OBJECTIVE FUNCTION
C      BETA        SCALAR USED TO SET STEP SIZE FOR STEEPEST ASCENT ROUTINE
C                  INITIALLY SET = 0.1, VALUE IS REDUCED UNTIL IT FALLS
C                  BELOW THE VALUE OF EPS (TERMINATION OF PROGRAM)
C      DELTA       PERTURBATIONS ADDED TO Z IN ORDER TO OBTAIN A TRIAL X
C      ALPHA       FRACTION OF RANGE USED TO EVALUATE PARTIAL DERIVATIVES
C      EPS,IDONE   TERMINATION CRITERION
C      IEXP        NUMBER OF EXPERIMENTS USED TO LOCATE THE HIGH POINT
C                  ALONG A VECTOR
C      FACT1       CONSTANT MULTIPLIER
C      FACT2       CONSTANT MULTIPLIER
C      IOUT        OUTPUT INDICATOR =1, INTERMEDIATE RESULTS PRINTED
C                  =0, ONLY BEGINNING, AND FINAL ANSWER PRINTED
C      ****************************************************************
C      SUBROUTINES REQUIRED
C      VECSCH      VECTOR SEARCH ALGORITHM
C      STPASC      STEEPEST ASCENT ALGORITHM
C      OBJFUN      USER SUPPLIED ROUTINE FOR THE OBJECTIVE FUNCTION
C                  MUST BE OF THE FORM      FUNCTION OBJFUN(X)
C                                           DIMENSION X( )
C                  FUNCTION STATEMENTS      -- -- -- -- -- -- -- --
C                  STATEMENT MUST BE LAST   OBJFUN=-- -- -- -- -- --
C                                           RETURN
C                                           END
C      WRITER      OUTPUT ROUTINE
C      BOUNDS      USER SUPPLIED ROUTINE TO ASSURE THAT PARTAN DOES NOT
C                  VIOLATE VARIABLE BOUNDS HAS FORM
C                                           SUBROUTINE BOUNDS(X)
C                                           DIMENSION X( )
C                  VARIABLE BOUNDARIES      -- -- -- -- -- -- -- --
C                                           RETURN
C                                           END
C                  CAN ALSO BE A DUMMY ROUTINE WITH ONLY THE DIMENSION,
C                  RETURN, AND END STATEMENTS
C      ****************************************************************
       LOGICAL IDONE
       DIMENSION  BC1(10), BC2(10)
       COMMON  X(10), Z(10), R(10), DELTA(10), N, FB, BETA, EPS, FACT1
```

```
      COMMON  FACT2, ALPHA, IEXP
C
C     INITIALIZE PROGRAM CONSTANTS AND FACTORS
C
      IDONE = .FALSE.
      BETA = .1
      FACT2 = .5+SQRT(5.)/2.
      FACT1 = 1.+FACT2
C
C     READ IN THE CONSTANTS FOR THE OBJECTIVE FUNCTION
C
      READ (5,1001) N, IEXP, EPS, ALPHA,IOUT
 1001 FORMAT (2I5,2E10.4,I5)
C
C     READ IN THE INITIAL VALUES FOR INDEPENDENT VARIABLES AND THEIR
C     RANGES
C
      DO 100 I=1,N
  100 READ (5,1002) Z(I), R(I)
 1002 FORMAT (2E10.4)
C
C     TRANSFER Z VECTOR INTO X AND BC1 VECTORS
C
      DO 105 I=1,N
      X(I) = Z(I)
  105 BC1(I) = Z(I)
C
C     EVALUATE FUNCTION AT INITIAL CONDITIONS,PRINT RESULTS,AND STORE
C     FUNCTION VALUE(F) INTO FB
C
      F = OBJFUN(X)
      WRITE (20,1004)
 1004 FORMAT(/2X,49HINITIAL STARTING CONDITION FOR OBJECTIVE FUNCTION)
      FB = F
      CALL WRITER (Z,N,FB,IOUT)
C
C     EVALUATE THE DELTAS FROM STPASC SUBROUTINE
C
      CALL STPASC
C
C     ESTABLISH A BASE POINT FOR THE VECTOR AND DETERMINE IF OPTIMALITY
C     HAS BEEN OBTAINED
C
      CALL VECSCH (IDONE,1)
      IF(IDONE) GO TO 999
C
C     START THE PARALLEL TANGENTS ALGORITHM
C
C
C     STORE LAST SUCCESSFUL TRIAL VALUES OF INDEPENDENT VARIABLES IN BC2
C
  110 DO 115 I=1,N
  115 BC2(I) = Z(I)
C
C     EVALUATE THE DELTAS FROM STPASC SUBROUTINE
C
```

```
      CALL STPASC
C
C     ESTABLISH A NEW POINT FOR THE VECTOR
C
      CALL VECSCH (IDONE,1)
      IF(IDONE) GO TO 999
C
C     ADJUST THE DELTAS SO ACCELERATION ALONG THE VECTOR CAN BE DONE
C
  120 DO 125 I=1,N
  125 DELTA(I) = (Z(I)-BC1(I))/3.
C
C     FIND THE HIGH POINT OF THE VECTOR AND DETERMINE IF OPTIMALITY HAS
C     BEEN REACHED
C
      CALL VECSCH (IDONE,0)
      IF(IDONE) GO TO 999
C
C     STORE THE LAST SUCCESSFUL VALUES OF THE INDEPENDENT VARIABLES (Z)
C     INTO BC1 FROM BC2   AND CONTINUE THE PARTAN ALGORITHM
C
  130 DO 135 I=1,N
  135 BC1(I) = BC2(I)
      FUNVAL=OBJFUN(Z)
      CALL WRITER(Z,N,FUNVAL,IOUT)
      GO TO 110
C
C     CAN ONLY REACH STATEMENT 999 IF THE VALUE OF BETA HAS BEEN REDUCED
C     BELOW THE USER VALUE OF EPS
C
  999 WRITE (20,1003)
 1003 FORMAT (/2X,21HOPTIMIZATION COMPLETE///)
      FUNVAL=OBJFUN(Z)
      CALL WRITER(Z,N,FUNVAL,IOUT)
      STOP
      END
C     *****************************************************************
      SUBROUTINE STPASC
C
C     STEEPEST ASCENT ALGORITHM
C
      DIMENSION  DELF(10)
      COMMON  X(10), Z(10), R(10), DELTA(10), N, FB, BETA, EPS, FACT1
      COMMON  FACT2, ALPHA, IEXP
C
C     EVALUATE THE DIRECTIONS OF STEEPEST ASCENT BY EVALUATING ALL
C     DELF(I)=F(X(1),....,X(I)+ALPHA*R(ANGE)(I),X(N))-F(Z)  WHERE ALPHA
C     IS A SMALL FRACTION (TYPICALLY 1.0E-03  TO 1.0E-06)
C
      DO 100 I=1,N
      X(I) = Z(I)+ALPHA*R(I)
      F = OBJFUN(X)
      X(I)=Z(I)
  100 DELF(I) = F-FB
C
C     DETERMINE THE LARGEST CHANGE IN THE VALUE OF THE OBJECTIVE FUNCTION
```

```
C
      AFMAX=ABS(DELF(1))
      DO 200 I=2,N
  200 AFMAX=AMAX1(AFMAX,ABS(DELF(I)))
C
C     INITIAL PERTURBATIONS ALONG THE VECTOR OF STEEPEST ASCENT ARE
C     EVALUATED FROM  DELTA(I)=BETA*R(ANGE)(I)*DELF(I)/(MAX ABSOLUTE VALUE
C     (DELF(I),I=1,N))
C     THE VARIABLE YIELDING THE MAXIMUM DELF(I) WILL BE PERTURBED A MAX
C     OF BETA*RANGE
C
      DO 300 I=1,N
  300 DELTA(I) = BETA*R(I)*DELF(I)/AFMAX
      RETURN
      END
C     ******************************************************************
      SUBROUTINE VECSCH (IDONE,KK)
      LOGICAL  IDONE
      COMMON  X(10), Z(10), R(10), DELTA(10), N, FB, BETA, EPS, FACT1
      COMMON  FACT2, ALPHA, IEXP
C
C     FROM HERE TO STATEMENT 145 THE ALGRORITHM IS DEVOTED TO BRACKETING
C     THE HIGH POINT ALONG A VECTOR
C
C
C     KK        INDICATOR VARIABLE IF KK=0 DONT REDUCE BETA THE STEP SIZE
C     KKK       INDICATOR VARIABLE FOR A GENERAL PROCEDURE INCORPORATED TO
C               YIELD GOOD GUESSES FOR DISTANCES TO BE TRIED ALONG THE VECTOR
C     KTEST     LOGICAL INDICATOR VARIABLE KTEST=1,SUCCESSFULLY BRACKETED
C               HIGH POINT KTEST=0 INCREASE STEP SIZE TRY AGAIN
C     S         INDICATOR VARIABLE S=1 EXPERIMENT RESULTED IN A SUCCESS
C               CONTINUE INCREASING INDEPENDENT VARIABLES
C               S=-1 EXPERIMENT RESULTED IN FAILURE, DECREASE VALUE OF
C               INDEPENDENT VARIABLES
C
      S = 0.
      KTEST=0
      J = 1
      KKK = -3
  100 DO 105 I=1,N
  105 X(I) = Z(I)+DELTA(I)
      CALL BOUNDS (X)
      F = OBJFUN(X)
      IF (F.GT.FB) GO TO 120
  110 DO 115 I=1,N
  115 DELTA(I) = DELTA(I)/FACT1
      KTEST=1
      IF (S) 145,135,145
  120 DO 125 I=1,N
      Z(I) = X(I)
  125 DELTA(I) = DELTA(I)*FACT2
      S = 1.
      F = FB
      IF (KTEST.EQ.0) GO TO 130
      GO TO 110
  130 IF (KK.EQ.0) GO TO 100
```

```
      BETA = BETA*FACT2
      GO TO 100
  135 IF (KK.EQ.0) GO TO 140
      BETA = BETA/FACT1
      IF (BETA.LT.EPS) GO TO 999
      GO TO 100
  140 KKK = KKK+1
      IF(KKK.LT.0) GO TO 100
C
C     FROM THIS POINT TO EXIT THE PROCEDURE IS TO ACCURATELY DETERMINE
C     THE HIGH POINT ALONG A VECTOR
C
C     EXPERIMENTS ARE PLACED IN THE INTERVAL CONTAINING THE HIGH POINT
C     TO MORE PRECISELY DETERMINE THE HIGH POINT
C
  145 DO 150 I=1,N
  150 X(I) = Z(I)+S*DELTA(I)
      CALL BOUNDS (X)
      F = OBJFUN(X)
      IF (F.GT.FB) GO TO 155
      S = -S
      GO TO 165
  155 FB = F
      DO 160 I=1,N
  160 Z(I) = X(I)
  165 IF (J.GT.IEXP) GO TO 175
      DO 170 I=1,N
  170 DELTA(I) = DELTA(I)/FACT2
      J = J+1
      IF (KKK.LT.0) GO TO 100
      GO TO 145
  175 DO 180 I=1,N
  180 X(I) = Z(I)
      RETURN
  999 IDONE = .TRUE.
      RETURN
      END
C     ******************************************************************
      SUBROUTINE WRITER(Z,N,FUNVAL,IOUT)
      DIMENSION  Z(1)
C
C     SUBROUTINE TO WRITE OUT THE VALUE OF THE OBJECTIVE FUNCTION AT ANY
C     POINT
C
      IF(IOUT.EQ.0) RETURN
      WRITE (20,100)
  100 FORMAT (///38HVALUE OF OBJECTIVE FUNCTION FOR THIS Z//)
      WRITE (20,200) (I,Z(I),I=1,N)
  200 FORMAT (2X5(2X2HZ(I2,4H) = E10.4))
      WRITE (20,300) FUNVAL
  300 FORMAT (//27HOBJECTIVE FUNCTION VALUE = F20.6//)
      RETURN
      END
C     ******************************************************************
      FUNCTION OBJFUN(W)
      DIMENSION W(1)
```

```
C     THIS FUNCTION ROUTINE DEFINES THE GEOMETRIC PROGRAMMING DUAL
C     FUNCTION OF SMITHS GENERALIZED PRODUCTION INVENTORY MODEL
C
C     ALL THE WEIGHTS  HAVE BEEN SOLVED IN TERMS OF W01,W02,W03,W04,W05,
C     AND W06 AND THESE VALUES ARE SUBSTITUTED INTO THE DUAL FUNCTION
C
      CALL BOUNDS (W)
      W01=W(1)
      W02=W(2)
      W03=W(3)
      W04=W(4)
      W05=1.-W01-W02-W03-W04-W(5)
      W06=W(5)
      W11=W01-W02
      W12=W03-W04
      W13=1.-(W01+W02+W03+W04+2.*W06)
      W10=W11+W12+W13
      A=((5./W01)**W01)
      A=A*((5.E4/W02)**W02)
      A=A*((20./W03)**W03)
      A=A*((7.2E4/W04)**W04)
      A=A*((10./W05)**W05)
      A=A*((1.44E5/W06)**W06)
      A=A*((4.*W10/W11)**W11)
      A=A*((32.*W10/W12)**W12)
      A=A*((120.*W10/W13)**W13)
  300 OBJFUN=A
      RETURN
      END
      SUBROUTINE BOUNDS(W)
      DIMENSION W(1)
C
C     SUBROUTINE TO CHECK THE BOUNDARY VALUES FOR THE INDEPENDENT
C     VARIABLES
C     1.  W(I) GT 0.0  IF W(I) NEGATIVE SET W(I)=1.0E-03
C     2.  W01 GT W02 IF NOT REDUCE W02
C     3.  W03 GT W04 IF NOT REDUCE W04
C
      DO 50 I=1,5
   50 IF(W(I).LE.0.) W(I)=1.E-3
      IF(W(1).GT.W(2)) GO TO 100
      W(2)=.8*W(1)
  100 IF(W(3).GT.W(4)) GO TO 200
      W(4)=.5*W(3)
  200 IF(J.EQ.0) RETURN
      RETURN
      END
```

APPENDIX B

PARALLEL TANGENTS PERFORMANCE

To test the performance ofthe P ARTAN computer program, the thesis committee submitted four problems (with degrees of difficulty greater than zero) to be optimized by means of geometric programming and the parallel tangents algorithm. Only one of these four problems proved to be unsolvable by the methods presented in Chapter Two. Further analysis and research is continuing to determine the cause for failure.

PROBLEM 1: (Unsolvable by PARTAN)

$$\text{Minimize: } C_T = 100{,}000(S-.1)^3 + 2100(1-S)^2 + 1095S^{-1}.$$

Simplification of the problem leads to the expression

$$\text{Minimize: } C_T' = C_T - 2000 = 100{,}000S^3 - 27{,}900S^2 - 1200S + 1095S^{-1}.$$

The geometric programming dual problem has two degrees of difficulty and the expression for the dual problem is

$$\text{Maximize: } \left(\frac{100{,}000}{w_{01}}\right)^{w_{01}} \left(\frac{27{,}900}{w_{02}}\right)^{-w_{02}} \left(\frac{1200}{w_{03}}\right)^{-w_{03}} \left(\frac{1095}{w_{04}}\right)^{w_{04}}$$

Subject to:

$$w_{01} - w_{02} - w_{03} + w_{04} = 1$$

$$3w_{01} - 2w_{02} - w_{03} - w_{04} = 0.$$

As stated before the problem proved to be unsolvable by means of PARTAN. All the combinations of two weights were attempted. Inability to solve this problem can more than likely attributable to the fact that bounds for the weights are unobtainable.

PROBLEM 2:

Minimize: $C_T = 10(0 + 1)P^{-1} + (22.5 + 5.73P)0^{-.5}$.

Simplification of the problem leads to the expression

Minimize: $C_T = 100P^{-1} + 10P^{-1} + 22.50^{-.5} + 5.73P0^{-1}$.

The geometric programming dual function of this problem has one degree of difficulty and the geometric programming equivalent of this problem is

$$\text{Maximize: } \left(\frac{10}{w_{01}}\right)^{w_{01}} \left(\frac{10}{w_{02}}\right)^{w_{02}} \left(\frac{22.5}{w_{03}}\right)^{w_{03}} \left(\frac{5.73}{w_{04}}\right)^{w_{04}} .$$

$$\begin{aligned} \text{Subject to: } \quad w_{01} + w_{02} + w_{03} + w_{04} &= 1 \\ -w_{01} - w_{02} + w_{04} &= 0 \\ w_{01} - .5w_{03} - .5w_{04} &= 0. \end{aligned}$$

By utilizing the techniques in Chapter Two, the most restrictive bounds are obtained for w_{04}, that is $1/3 \leq w_{04} \leq 1/2$. By solving the constraint set in terms of w_{04} and using PARTAN to seek theoptimum, the minimum cost of 34.778 is obtained when w_{04} = 0.3682. Values for the primal variable are P=5.4978, and θ = 6.0236. Using these values in the expressions for the derivative of the primal cost function with respect to P and θ forces the derivatives to vanish. Thus PARTAN was able to obtain the optimum.

PROBLEM 3:

$$\text{Minimize: } C_T = 5000Q_B^{.7} + (324)(3800)Q_B^{-1} + 2000\left(\frac{.12 \times 2.12}{2} + .01\right)Q_b.$$

This problem has one degree of difficulty and the geometric programming dual function is

$$\text{Maximize: } \left(\frac{5000}{w_{01}}\right)^{w_{01}} \left(\frac{1.2312\text{x}10^6}{w_{02}}\right)^{w_{02}} \left(\frac{274.4}{w_{03}}\right)^{w_{03}}$$

$$\text{Subject to: } w_{01} + w_{02} + w_{03} = 1$$

$$.7w_{01} - w_{02} + w_{03} = 0.$$

Inspection of the above constraint set shows that the most restrictive bounds that can be obtained are for w_{02} ($.412 \leq w_{02} \leq .5$). Using PARTAN the optimum cost is 103,175.0 at w_{02}=.4299. Assuming that PARTAN obtained the optimum, the optimum value of Q_B is 28.08455. Since the expression for the derivative of C_T with respect to Q_b is not directly solvable, substitution of the value for Q_B into the expression for the derivative does force it to vanish. Therefore PARTAN did obtain an optimum for the problem.

PROBLEM 4:

$$\text{Minimize: } \frac{10Q_B^{1.2}}{Q_A} + 465Q_B^{-1} + 216(Q_B^{-.78} + 1.25Q_B^{-1})$$

$+ Q_A x 10^{-6}$.

Simplification of the above expression leads to the two degree of difficulty primal problem

$$\text{Minimize: } 10Q_A^{-1}Q_B^{1.2} + 456Q_B^{-1} + 21.6x1.25Q_B^{-1} + 10^{-6}Q_A.$$

The geometric programming dual problem is

$$\text{Maximize: } \left(\frac{10}{w_{01}}\right)^{w_{01}} \left(\frac{456}{w_{02}}\right)^{w_{02}} \left(\frac{21.6}{w_{03}}\right)^{w_{03}} \left(\frac{21.6x1.25}{w_{04}}\right)^{w_{04}} \left(\frac{10^{-6}}{w_{05}}\right)^{w_{05}}.$$

$$\begin{aligned} \text{Subject to: } & w_{01} + w_{02} + w_{03} + w_{04} + w_{05} = 1 \\ & -w_{01} + w_{05} = 0 \\ & 1.2w_{01} - w_{02} - .78w_{03} - w_{04} = 0. \end{aligned}$$

By experimentation the most restrictive bounds that can be obtained are for w_{01} and w_{02} ($.78/3.2 \leq w_{01} \leq 1/3.2$ and $0 \leq w_{02} \leq .294/.407$). Using these bounds in the PARTAN program, the optimum cost is found to be 0.898. Solving for Q_A and Q_B and assuming this is the optimum cost, the values for Q_B and Q_A are 65,309.6 and 1717.8 respectively. By substitution of these values into the expressions for the derivatives of the cost function with respect to Q_A and Q_B forces these expressions to vanish. Therefore optimility has been determined by the parallel tangents algorithm.