A GENERALIZED ALGORITHM USING THE HARMONIC MEAN FOR SOLVING UNCONSTRAINED BALANCED POSYNOMIALS

by
Mark B. Pomeroy

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 (Mathematical and Computer Sciences).

Golden, Colorado

Date 5 OCT 1995

Signed: Mark B. Pomeroy
Mark B. Pomeroy

Approved: R. E. D. Woolsey
Dr. R. E. D. Woolsey
Thesis Advisor

Golden, Colorado

Date 5 OCT 1995

Graeme Fairweather
Graeme Fairweather
Professor and Head
Mathematical and Computer Sciences Department

ABSTRACT

A generalized algorithm, called harmonic programming, which is based on the harmonic mean, will solve a large class of unconstrained nonlinear optimization problems which have balanced exponents. This algorithm is then expanded by using a technique similar to that used by Ratliff in geometric programming, to solve multivariable, multiple degree of difficulty problems in the form described above.

The new algorithm opens the door to a whole new field in nonlinear optimization problem solving. The algorithm covers a large span of nonlinear optimization problems in both engineering, and economics. In addition, harmonic programming was for every test problem, as good, or (in most cases) better in running time and accuracy than MINOS, LINGO, and MULTICON.

The algorithm was successfully tested on a variety of engineering, economic and nonlinear test problems. Overall, harmonic programming appears to have the same general applicability as geometric programming.

TABLE OF CONTENTS

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank Dr. Woolsey for serving as my advisor, and for introducing me to Geometric Programming, which eventually led to my thesis topic. Additionally, my thanks go to Dr. Maurer and Professor Astle for assisting in my graduate education, and respectively, for chairing and serving on my thesis committee.

Several fellow students deserve recognition for their assistance with this thesis. I thank CPT Bill Dolan for helping me to get my algorithm to work for the first test problem, and to Jason Kierstein for his expertise in turning my algorithm into a program in FORTRAN.

I thank my wife, Kelly, for all of her support throughout my Army and academic careers.

DEDICATION

I would like to dedicate this thesis, first, to my beautiful, loving bride and best friend, Kelly, whose devotion to our family and my military career have never wavered. Secondly, I would like to dedicate this to my two sons, Luke and Kyle, who make every day an adventure.

I want also to dedicate this thesis to my parents who have made me who I am today.

Chapter 1

INTRODUCTION

1.1 The Arithmetic-Geometric-Harmonic Mean Inequality and Posynomial Functions

The arithmetic-geometric mean inequality is the foundation for geometric programming. In a similar manner, the arithmetic-geometric-harmonic mean (A.M. - G.M. - H.M.) inequality is the foundation for harmonic programming. This inequality can be represented as follows

$$\underset{\text{A.M.}}{\frac{1}{n}\sum_{i=1}^{n} x_i} \geq \underset{\text{G.M.}}{\sqrt[n]{x_1 \cdot x_2 \cdots x_n}} \geq \underset{\text{H.M.}}{\frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}} \tag{1.1}$$

Using weighted means it can be rewritten in the form (Duffin, Peterson and Zener 1967 pg. 315)

$$\underset{\text{A.M.}}{\sum_{i=1}^{n} \alpha_i v_i} \geq \underset{\text{G.M.}}{\prod_{i=1}^{n} v_i^{\alpha_i}} \geq \underset{\text{H.M.}}{\left[\sum_{i=1}^{n} \frac{\alpha_i}{v_i}\right]^{-1}} \tag{1.2}$$

where the v_i are positive quantities and α_i are nonnegative weights which must sum to one. Letting $u_i = \alpha_i v_i$ yields

$$\underset{\text{A.M.}}{\sum_{i=1}^{n} u_i} \geq \underset{\text{G.M.}}{\prod_{i=1}^{n}\left(\frac{u_i}{\alpha_i}\right)^{\alpha_i}} \geq \underset{\text{H.M.}}{\left[\sum_{i=1}^{n}\left(\frac{\alpha_i^2}{u_i}\right)\right]^{-1}} \tag{1.3}$$

The variables (u_i) are positive quantities, and the inequalities hold if and only if

$$u_i = \alpha_i \sum_{j=1}^{n} u_j \,; \qquad \text{for } i = 1, 2, \ldots, n. \tag{1.4}$$

The harmonic programming algorithm described in chapters 2 and 3 will use the inequality described above, and is designed to solve unconstrained nonlinear optimization problems in the form

$$\text{Minimize} \quad z = \sum_{i=1}^{n} K_i \prod_{j=1}^{m} x_j^{a_{ij}} \,; \tag{1.5}$$

where

$$\begin{aligned} &\boldsymbol{K_i} > \mathbf{0}, \\ &\boldsymbol{a_{ij}} \in \Re, \\ &\boldsymbol{x_j} \in \Re^{+} \\ &\Re^{+} = \text{Positive real numbers} \end{aligned}$$

for $i = 1, \ldots, n; \;\; j = 1, \ldots, m.$

When the coefficients (K_i) have a positive value, the problem in the form listed is called a posynomial function.

It is assumed here, that the reader has a general knowledge of geometric programming. Some of the areas of geometric programming which are used in harmonic programming will be discussed briefly in the following sections.

1.2 The Nonnegative Weights (α_i's)

The nonnegative weights (α_i's) are the percentage contributions of each term to the objective function. For the remainder of this thesis, when the optimal weight of each term (α_i), is discussed it will be called delta (δ_i). These weights, as with geometric programming, play an integral part in harmonic programming. The contribution of each term remains the same regardless of whether the arithmetic, geometric, or harmonic mean is used.

Most of the early work in geometric programming was done by Duffin, Peterson, and Zener (1967). Dr. R. E. D. Woolsey used their concepts to develop four rules to solve zero degree of difficulty geometric programming problems (Woolsey 1992). Since the δ_i's remain the same for harmonic programming, two of his rules, which pertain to the deltas, will be used extensively (rule II and rule III). Rule II is used to solve for the δ_i's. The δ_i's must satisfy two conditions. The first condition, which was stated above, will be referred to as the normality condition, where

$$\sum_{i=1}^{n} \delta_i = 1. \tag{1.6}$$

The second condition, which will be called the orthogonality condition, requires the following

$$D_j = \sum_{i=1}^{n} a_{ij}\delta_i = 0; \qquad \text{for } j = 1, 2, ..., m \tag{1.7}$$

where a_{ij} is the power for term_i and variable_j. The final condition requires that

$$\delta_i > 0, \text{ for } i = 1, 2, ..., n \tag{1.8}$$

For zero degree of difficulty problems, these conditions can be written as a system of simultaneous equations called the exponent matrix, and then solved for the δ_i's. For example, given the following optimization problem:

$$\text{Minimize TC} = 1.43x^{-1} + 1656x^{-1}s^{-1} + 47.6x^{.9}s^{.36},$$

the exponent matrix is

$$\begin{aligned} &\delta_1 + \delta_2 + \delta_3 = 1 \\ &-\delta_1 - \delta_2 + .9\delta_3 = 0 \\ &0\delta_1 - \delta_2 + .36\delta_3 = 0 \end{aligned}$$

Solving the system of equations yields $\delta_1 = .284$, $\delta_2 = .189$, $\delta_3 = .526$.

The second of Woolsey's rules which will be used in harmonic programming is rule III. Rule III uses the δ_i's to back out the values for each variable from the optimal value of the objective function. This can be written as:

$$z^* = \frac{FIRST\_TERM\_OF\_OBJ.\_FUN.}{\delta_1} = \ldots = \frac{NTH\_TERM\_OF\_OBJ.\_FUN.}{\delta_{nth}} \tag{1.9}$$

where z* is the value of the objective function at optimality.

1.3 Condensation and Ratliff's Method

Condensation is a method developed by Duffin, Peterson, and Zener (1967), in which, as described by Beightler and Phillips (1976, pp. 331-367), a multiterm posynomial function is approximated with a monomial or a single term function. The primary advantage of this technique is that the number of degrees of difficulty of the problem can be reduced without reducing the number of variables. A single variable problem can be reduced to a zero degree of difficulty problem by condensing the terms with positive and negative exponents separately, and then restating the objective function. A synopsis of condensation for a single variable problem follows. Given a single variable posynomial function in the form

$$z = \sum_{i=1}^{n} K_i x^{a_i} + \sum_{j=1}^{n} L_j x^{-b_j}; \tag{1.10}$$

using the arithmetic-geometric mean inequality, the terms with positive powers in the function can be restated as

$$y = \sum_{i=1}^{n} K_i x^{a_i} \geq \prod_{i=1}^{n} \left(\frac{K_i}{\alpha_i} \right)^{\alpha_i} \times x^{\sum a_i \times \alpha_i}; \tag{1.11}$$

where the alphas can be defined as

$$\alpha_i = \left(\frac{K_i x^{a_i}}{\sum_{i=1}^{n} K_i x^{a_i}} \right). \tag{1.12}$$

In a like manner, the terms with negative powers can be condensed. Adding the condensed positively powered terms and the condensed negatively powered terms yields the following inequality, where the right hand side is now a zero degree of difficulty problem

$$z = \sum_{i=1}^{n} K_i x^{a_i} + \sum_{j=1}^{m} L_j x^{-b_j} \geq \prod_{i=1}^{n} \left(\frac{K_i}{\alpha_i} \right)^{\alpha_i} \times x^{\sum_{i=1}^{n} a_i \times \alpha_i} + \prod_{j=1}^{m} \left(\frac{L_j}{\alpha_j} \right)^{\alpha_j} \times x^{\sum_{j=1}^{m} b_j \times \alpha_j}; \quad (1.13)$$

Using this approach, Richard M. Ratliff developed the MULTICON algorithm in 1986. MULTICON is a generalized condensation algorithm for the solution of unconstrained, balanced, multivariable, posynomial problems using geometric programming.

A brief version of his algorithm (Ratliff 1986) follows:

1) Put the equation in unconstrained, balanced posynomial form.

2) Choose initial values for each variable in the problem, and call these variables x_{iold} (where i = 1 to the number of variables). Treat all variables but one as constants using the values of x_{iold}. State the simplified single variable problem with revised coefficients.

3) Condense the simplified objective function into a zero degree difficulty problem. Solve the problem using conventional geometric programming techniques. Extract a new value for variable of interest (x_{inew}).

4) Using x_{inew}, calculate a value for the simplified objective function (VALHAT). Compare values of VALHAT on successive iterations. When the difference becomes negligible for all variables, use the current variable values as the final solution.

5) If the difference is not negligible set $x_{iold} = x_{inew}$. Treat the next variable in the original objective function as a variable, and all others as constants. State the simplified single variable problem with revised coefficients. Return to step 3, and continue stepping through the algorithm until changes in the objective function become negligible.

A detailed discussion of MULTICON can be found in Ratliff's thesis (1986), and condensation can be referenced in Beightler and Phillips (1976, pp. 331-367), and Woolsey (1992, pp. 3-1 through 3-6).

1.4 Previous Uses of the Harmonic Mean

In the past, the harmonic mean has rarely been used in mathematical programming. The most significant use of the harmonic mean in solving optimization type problems is a method developed by Duffin and Peterson in 1972, and later described by Beightler and Phillips in 1976. This method is called "treating reversed geometric programs with harmonic means." A simplified version of Beightler and Phillips' description is given in the subsequent three paragraphs.

Geometric programming is designed to solve posynomial minimization problems in the form shown in equation (1.5) containing only prototype constraints. A prototype constraint is one in the form

$$y_m(x) \le 1, \tag{1.14}$$

where $y_m(x)$ is a posynomial. A reversed geometric program is one which contains one or more reversed constraints in the form

$$y_m(x) \ge 1, \tag{1.15}$$

where again $y_m(x)$ is a posynomial. Each reversed constraint in the form of equation (1.15) can then be converted into a prototype constraint in the form

$$\frac{1}{y_m(x)} \le 1. \tag{1.16}$$

Calling each term of $y_m(x)$, u_i, equation (1.16) can be rewritten as

$$\left[\sum_{i=1}^{m} u_i\right]^{-1} \le 1; \qquad m = 1, 2, \ldots, M, \tag{1.17}$$

where M is the number of terms.

When the reversed constraint (1.15) is converted into a prototype constraint (1.17), it can be difficult to work with computationally. This constraint can be further restated using either the geometric mean approximation or the harmonic mean approximation. The geometric mean approximation is most useful when it is desirable to reduce the constraint into one term and, as a result, reduce the degrees of difficulty of the entire problem. The harmonic mean approximation, on the other hand, is most useful when using the geometric mean approximation would reduce the degrees of difficulty of the problem below zero. The geometric mean approximation for equation (1.17) is

$$\prod_{i=1}^{m}\left[\frac{u_i}{\alpha_i}\right]^{-\alpha_i} \leq 1, \tag{1.18}$$

where α_i is the weight associated with each term. The harmonic mean approximation for equation (1.17) is

$$\sum_{i=1}^{m}\left[\frac{\alpha_i^2}{u_i}\right] \leq 1. \tag{1.19}$$

A brief version of Beightler and Phillips' algorithm (1976) follows:

1) Put the equation in constrained, balanced posynomial form. Pick a feasible solution for each variable.

2) Approximate the reversed constraint by either the harmonic or geometric mean. Calculate the weight for each term in the reversed constraint using equation (1.12), and the values picked in step 1 for the first iteration and those from step 3 for subsequent iterations.

3) Solve the restated problem using a posynomial programming code.

4) Using the solution from step 3, determine whether or not the original constraints are satisfied. If they are satisfied, stop. If not, return to step 2.

1.5 Chapter 1 Summary

In this chapter, the arithmetic-geometric-harmonic mean inequality, posynomial functions, the nonnegative weights, condensation, Ratliff's method, and previous uses of the harmonic mean were addressed. These topics are the fundamental concepts which were used to develop the harmonic programming algorithm. Chapter 2 covers the development of the three harmonic programming algorithms, and finally, the general algorithm for harmonic programming.

Chapter 2

THE HARMONIC PROGRAMMING ALGORITHM

2.1 General

As stated in chapter 1, the arithmetic-geometric-mean inequality is the foundation for the development of a harmonic programming algorithm. The algorithm is designed to solve unconstrained nonlinear optimization problems in the form

$$\text{Minimize } z = \sum_{i=1}^{n} K_i \prod_{j=1}^{m} x_j^{a_{ij}}; \tag{2.1}$$

where

$K_i \rangle 0$,
$a_{ij} \in \Re$,
$x_j \in \Re^+$
$\Re^+ =$ Positive real numbers

for i= 1, ...,n; j= 1, ...,m.

When all the coefficients (K_i) have a positive value, the problem in the form listed above is called a posynomial function. When the coefficients have negative values the problem is referred to as a signomial function. Although signomials are touched upon in this thesis, the primary algorithm is designed to solve posynomials. The first algorithm focuses on zero degree of difficulty problems and subsequent algorithms solve multiple degree of difficulty problems.

2.2 Harmonic Programming Algorithm #1: An Algorithm to Solve Unconstrained, Zero Degree of Difficulty, Posynomial Optimization Problems in the Form of Equation (2.1)

From the weighted arithmetic-geometric-harmonic mean inequality

$$\sum_{i=1}^{n} u_i \geq \prod_{i=1}^{n}\left(\frac{u_i}{\delta_i}\right)^{\alpha_i} \geq \left[\sum_{i=1}^{n}\left(\frac{\delta_i^2}{u_i}\right)\right]^{-1}, \tag{2.2}$$

A.M. G.M. H.M.

the variables u_i are, positive quantities, and the inequalities hold at equality if and only if

$$u_i = \omega_i \sum_{j=1}^{n} u_j ; \qquad \text{for } i = 1, 2, \ldots, n$$

(2.3)

(Beightler and Phillipps 1976 pg.315). For the first algorithm, only the arithmetic-harmonic-mean inequality will be used. The following inductive proof will show that that if

$$\begin{aligned} &\delta_i = \frac{u_i}{\sum_{i=1}^{n} u_i}; \\ &\forall u_i \geq 0; \\ &u_i \in \Re, \end{aligned} \tag{2.4}$$

then the arithmetic-harmonic mean inequality will become an equality (2.5).

$$\sum_{i=1}^{n} u_i = \left[\sum_{i=1}^{n}\left(\frac{\delta_i^2}{u_i}\right)\right]^{-1} \tag{2.5}$$

This proof is fundamentally important to harmonic programming. It is important to note, that the reverse is also true, specifically, that if the inequality is an equality, then the conditions in (2.4) will also be true. Since we are primarily concerned with the first proof this is all that will be shown.

Proof 2.2.1

Given that the one and two term examples are trivial, we begin the proof with a three term example by stating the arithmetic-harmonic-mean portion of equation (2.2).

$$u_1 + u_2 + u_3 \geq \left[\frac{\delta_1^2}{u_1} + \frac{\delta_2^2}{u_2} + \frac{\delta_3^2}{u_3}\right]^{-1} \tag{2.6}$$

Assuming that

$$\delta_i = \frac{u_i}{\sum_{i=1}^{n} u_i}, \tag{2.7}$$

we can substitute this into (2.6), which yields

$$u_1 + u_2 + u_3 \geq \left[\frac{\left(\frac{u_1}{u_1+u_2+u_3}\right)^2}{u_1} + \frac{\left(\frac{u_2}{u_1+u_2+u_3}\right)^2}{u_2} + \frac{\left(\frac{u_3}{u_1+u_2+u_3}\right)^2}{u_3}\right]^{-1}. \tag{2.8}$$

Simplifying inside the brackets in (2.8) yields the following inequality

$$u_1 + u_2 + u_3 \geq \left[\frac{u_1}{(u_1 + u_2 + u_3)^2} + \frac{u_2}{(u_1 + u_2 + u_3)^2} + \frac{u_3}{(u_1 + u_2 + u_3)^2} \right]^{-1}. \tag{2.9}$$

Since the denominators of each term of the harmonic mean approximation in (2.9) are now the same, it follows that

$$u_1 + u_2 + u_3 \geq \left[\frac{(u_1 + u_2 + u_3)}{(u_1 + u_2 + u_3)^2} \right]^{-1}. \tag{2.10}$$

Dividing the numerator and denominator of the harmonic mean approximation in (2.10) by $(u_1 + u_2 + u_3)$, yields

$$u_1 + u_2 + u_3 \geq \left[\frac{1}{(u_1 + u_2 + u_3)} \right]^{-1}, \tag{2.11}$$

which can be restated as

$$u_1 + u_2 + u_3 = u_1 + u_2 + u_3. \tag{2.12}$$

Thus for this three term example (n = 3) we see that, if the deltas equal the weights for each term, the inequality becomes an equality. Assuming that this argument is true for the case n=k, we must now show that this implies it is true for the case n=k+1, to complete the inductive argument. Assuming the argument is true for the n=k case, we begin with the following equality

$$u_1 + \ldots + u_k = \left[\frac{\delta_1^2}{u_1} + \ldots + \frac{\delta_k^2}{u_k} \right]^{-1}. \tag{2.13}$$

Adding u_{k+1} to both sides of equation (2.13) yields

$$u_1+\ldots+u_k+u_{k+1}=\left[\frac{\delta_1^2}{u_1}+\ldots+\frac{\delta_k^2}{u_k}\right]^{-1}+u_{k+1}. \tag{2.14}$$

Substituting

$$\delta_k=\frac{u_k}{\sum_{k=1}^{n}u_k} \tag{2.15}$$

into (2.14) yields

$$u_1+\ldots+u_k+u_{k+1}=\left[\frac{\left(\frac{u_1}{u_1+\ldots+u_k}\right)^2}{u_1}+\ldots+\frac{\left(\frac{u_k}{u_1+\ldots+u_k}\right)^2}{u_k}\right]^{-1}+u_{k+1}. \tag{2.16}$$

Simplifying inside the brackets in (2.16) yields the following equality

$$u_1+\ldots+u_k+u_{k+1}=\left[\frac{u_1}{(u_1+\ldots+u_k)^2}+\ldots+\frac{u_k}{(u_1+\ldots+u_k)^2}\right]^{-1}+u_{k+1}. \tag{2.17}$$

Since the denominators of each term of the harmonic mean approximation in (2.17) are now the same, it follows that

$$u_1+\ldots+u_k+u_{k+1}=\left[\frac{u_1+\ldots+u_k}{(u_1+\ldots+u_k)^2}\right]^{-1}+u_{k+1}. \tag{2.18}$$

Dividing the numerator and denominator of the harmonic mean approximation in (2.18) by $(u_1+\ldots+u_k)$, yields

$$u_1+\ldots+u_k+u_{k+1}=\left[\frac{1}{(u_1+\ldots+u_k)}\right]^{-1}+u_{k+1}. \tag{2.19}$$

which can be restated as

$$u_1+\ldots+u_k + u_{k+1} = u_1+\ldots+u_k + u_{k+1}. \tag{2.20}$$

Thus we have shown by induction, that if the deltas equal the weights for each term, the inequality becomes an equality. It is important to note that, if the optimal deltas are not chosen, the inequality will still become an equality when the weights equal the deltas. Since zero degree of difficulty, balanced, posynomials are globally optimal (Beightler and Phillips 1976 pg. 115), there is only one optimal solution. Therefore, if the optimal deltas are used, the x_i 's will converge to optimality. As previously stated, for zero degree of difficulty posynomial problems, the optimal δ_i's can be calculated for each term of a problem, in the form of equation (2.1), using Woolsey's rule II. For example, given the following unconstrained, zero degree of difficulty optimization problem:

$$\text{Minimize TC} = 1.43x^{-1} + 1656x^{-1}s^{-1} + 47.6x^{.9}s^{.36}, \tag{2.21}$$

as shown in chapter 1, the exponent matrix is

$$\begin{aligned}
&\delta_1 + \delta_2 + \delta_3 = 1 \\
&-\delta_1 - \delta_2 + .9\delta_3 = 0 \\
&0\delta_1 - \delta_2 + .36\delta_3 = 0.
\end{aligned}$$

Solving the system of equations yields: $\delta_1 = .284$, $\delta_2 = .189$, $\delta_3 = .526$. Letting $u_i = \text{term}_i$, and substituting these values into the harmonic and arithmetic portions of equation (2.2) yields

$$\text{Arithmetic Mean:} \quad 1.43x^{-1} + 1656x^{-1}s^{-1} + 47.6x^{.9}s^{.36} \geq \tag{2.22}$$

$$\text{Harmonic Mean: } \left[\frac{.284^2}{1.43x^{-1}} + \frac{.189^2}{1656x^{-1}s^{-1}} + \frac{.526^2}{47.6x^{.9}s^{.36}}\right]^{-1}. \tag{2.23}$$

Using the optimal weights, if values for each of the variables are chosen at random, we know that the inequalities will not become equalities unless optimality has been reached. Choosing x = s = 1, and substituting these values into (2.22), and (2.23), gives arithmetic and harmonic mean approximations which will henceforth be called z_{obj} and z_h, respectively, of 1705 and 16.07. Using the harmonic mean approximation (z_h) and the optimal deltas, new values for x and s can then be backed out, using Woolsey's rule III which was described in chapter 1. This method very closely resembles the method Ratliff used with geometric programming. The new values for each variable will be closer to optimality than the old values. The calculations are as follows

$$\begin{aligned}
&16.07 = \frac{1.43x^{-1}}{.284};\\
&\therefore x_{new} = .313;\\
&\frac{1.43x^{-1}}{.284} = \frac{1656x^{-1}s^{-1}}{.189};\\
&\therefore s_{new} = 1740.
\end{aligned} \tag{2.24}$$

Using these new values for x and s, in equations (2.22), and (2.23), gives $z_{objnew} = 253.19$, and $z_{hnew} = 32.75$. Extensive computational experience suggests that, with successive iterations, the values for each variable will eventually converge, as will the values of z_{obj}

and z_h. For this example, it is apparent that neither the variables, nor the mean approximations have converged yet.

Labeling $x_{old} = x_{new}$, $s_{old} = s_{new}$, and using z_{hnew}, the second iteration begins. This process continues for seven iterations using epsilon = .01. The results are as follows

$z_{obj} = 94.6$

$z_h = 94.6$

$x = .0532$

$s = 1740.$

Comparing these results to the known values at optimality, of z = 94.65, s = 1740, and x = .0532, it is apparent that optimality has been reached for this problem using harmonic programming.

In equation (2.24) rule III was used to calculate new values for x and s. Although this is easy to do by hand, it is significantly harder to program when a variable does not exist by itself in a term (e.g., s above). To simplify the programming, the following technique was used.

1) Does the variable exist by itself in a term? If yes, then solve for its new value using rule III, and move to the next variable. Check and solve for all variables that exist by themselves.

For example: Does x appear by itself in a term? Yes, then

$$\begin{aligned} 16.07 &= \frac{1.43x^{-1}}{.284}; \\ x_{new} &= .313 \end{aligned} \tag{2.25}$$

The next variable is s. Does s appear by itself in a term? No. Have all other variables been checked? Yes, then go to step two.

2) Start with the first variable that does not exist by itself in a term. Call this variable x_{int}. Use the latest value calculated for all other variables, and set them as constants. State the simplified objective function ignoring any constants.

For example: s is the first variable that does not exist by itself in a term; $s = x_{int}$. Use the latest value calculated for all other variables, and set them as constants, i.e. x = .313. State the simplified objective function

$$\begin{aligned} z_{simplified} &= \frac{1656}{.313}s^{-1} + 47.6(.313)^{.9}s^{.3} \\ &= 5290.7s^{-1} + 16.7s^{.3}. \end{aligned} \tag{2.26}$$

3) Is the simplified objective function for x_{int} a zero degree of difficulty problem? If yes, then using the latest value for x_{int}, calculate new deltas, and the harmonic mean approximation. If no, go to step five.

For example: is the simplified objective function (2.26) zero degree of difficulty? Yes. Using rule II: $\delta_1 = .23$ and $\delta_2 = .77$. The harmonic mean approximation is

$$\begin{aligned} z_h &= \left[\frac{.23^2}{5290.7(1)^{-1}} + \frac{.77^2}{16.7(1)^{.3}}\right]^{-1} \\ &= 28.15 \end{aligned} \tag{2.27}$$

4) Solve for a new value of x_{int} using rule III, and the deltas and harmonic mean approximation calculated in step three. Move to the next variable that does not exist alone in a term, and go to step two. Continue until new values have been calculated for each variable.

For example: using rule III

$$\begin{aligned} &28.15 = (5290s^{-1})/.23; \\ &\therefore s_{new} = 816.9. \end{aligned} \tag{2.28}$$

5) Condense the simplified objective function into a zero degree of difficulty problem using the method outlined in chapter 1. Go to step 3.

For example: if the simplified objective function (2.26), had instead been the following one degree of difficulty problem

$$z_{simplified} = 5290.7s^{-1} + 16.7s^{3} + 2s^{2}, \tag{2.29}$$

it would have needed to be condensed, before solving for the new value, of the variable of interest. Using the method outlined in chapter 1, this problem can be condensed in the following manner

a. Group the positively powered terms and negatively powered terms together. For this problem, since there is only one negatively powered term, it does not need to be condensed. The positively powered terms are

$$16.7s^{3} + 2s^{2}. \tag{2.30}$$

b. Calculate the weights for each term using the latest value for the variable of interest.

$$\begin{aligned} &\omega_1 = \frac{16.7(1)^{.3}}{16.7(1)^{.3} + 2(1)^{.2}}, \\ &\therefore \omega_1 = .89; \\ &\omega_2 = \frac{2(1)^{.2}}{16.7(1)^{.3} + 2(1)^{.2}}, \\ &\therefore \omega_2 = .11. \end{aligned} \tag{2.31}$$

c. Using the weights calculated above, condense the terms.

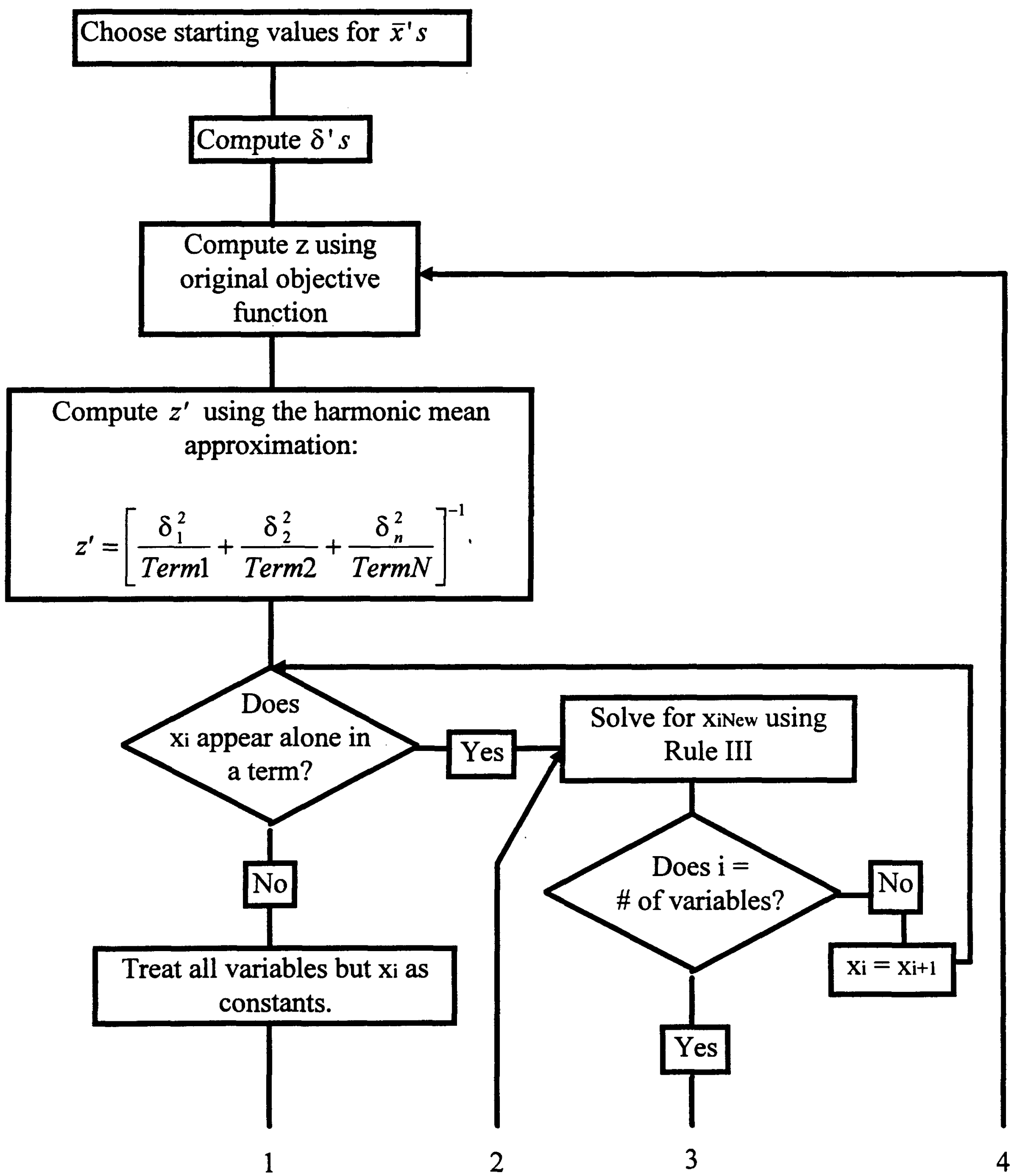
$$\begin{aligned} &\left(\frac{16.7s^{.3}}{.89}\right)^{.89} \times \left(\frac{2s^{2}}{.11}\right)^{.11}, \\ &= 18.62s^{.487} \end{aligned} \tag{2.32}$$

d. Combine the condensed positively powered term and negatively powered term and state the new zero degree of difficulty, simplified objective function.

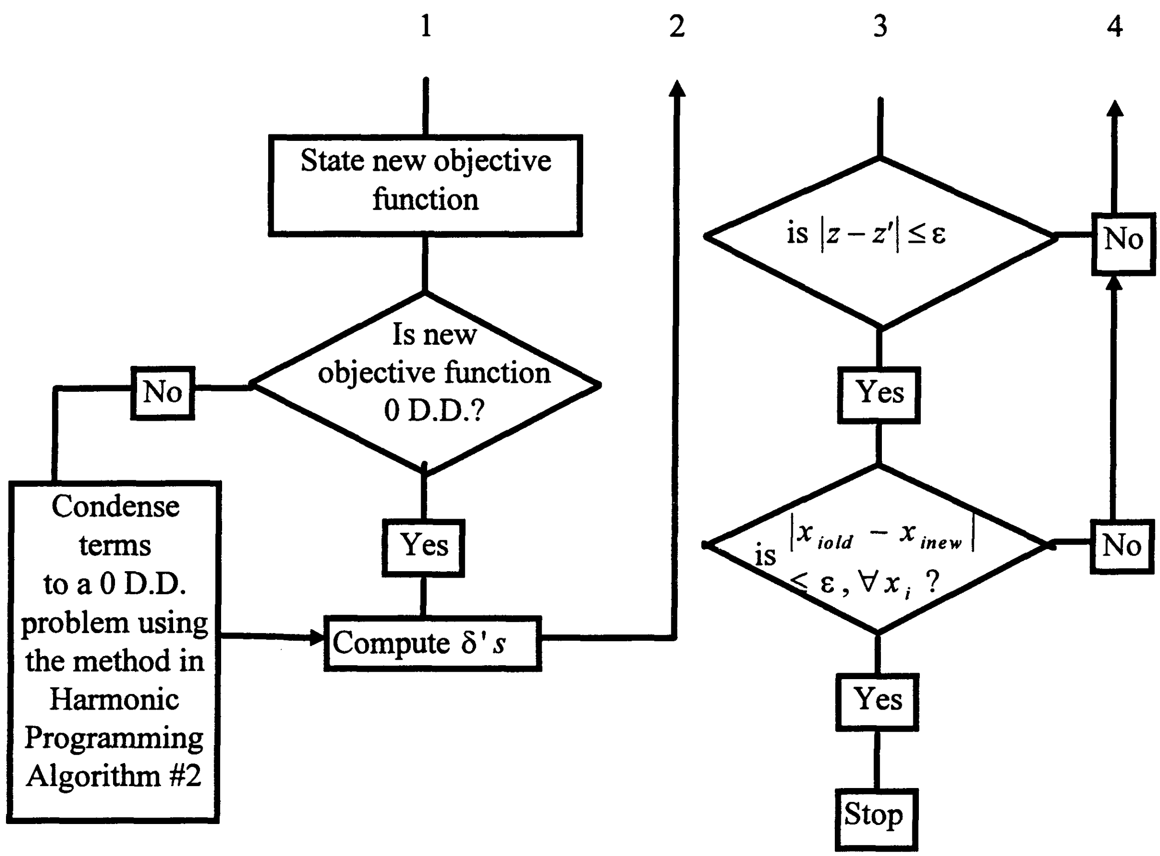
$$z_{simplified} = 5290.7s^{-1} + 18.62s^{.487} \tag{2.33}$$

The algorithm used in this section is called Harmonic Programming Algorithm #1. As mentioned before, this algorithm is designed to solve unconstrained, zero degree of difficulty, posynomial optimization problems in the form of equation (2.1). This algorithm is the basis for the other algorithms which will be described in the next section. The flow chart for this algorithm follows on the next page.

Harmonic Programming Algorithm #1 Flowchart



Harmonic Programming Algorithm #1 Flowchart (continued):



2.3 Harmonic Programming Algorithm #2: An Algorithm to Solve Unconstrained, Multiple Degree of Difficulty, Single Variable, Posynomial Optimization Problems in the Form of Equation (2.1)

This algorithm uses condensation extensively. It is designed to solve unconstrained, multiple degree of difficulty, single variable, posynomial optimization problems in the form of equation (2.1). The approach is to condense the problem into a zero degree of difficulty problem, solve the simplified problem, and back out a new variable using the method described in harmonic programming algorithm #1. The new value is then used to recondense the original problem. This process is repeated, until the values of the variable converge, between successive iterations. This technique is very similar to the approach Ratliff used in MULTICON, with the exception that the harmonic mean approximation is used in place of the geometric mean approximation. This algorithm, along with the one described in section 2.2, will be combined to give the third algorithm which solves multivariable, multiple degree of difficulty problems. Since there are no new concepts introduced for this algorithm, a step by step example follows.

The economic order quantity model for use in nuclear medicine as reported by Woolsey (1992), is a simple example of a problem which can be solved using harmonic programming algorithm #2. The problem is

$$\text{Minimize:} \quad Cost = 10Q + 1000Q^{-1} + Q^2 \tag{2.34}$$

1) Group together negatively powered terms and positively powered terms. Since there is only one negatively powered term, it does not need to be condensed. The following condensation steps will address only the positively powered terms. If there had been more than one negatively powered term, the same approach would be used to condense them. The positively powered terms are

$$10Q + Q^2 \tag{2.35}$$

2) Pick a starting value for x. Call this value xbar. For this example xbar = 1.

3) Using xbar, calculate the condensation weights for each term.

$$\begin{aligned} \omega_1 &= \frac{10(1)}{10(1)+1^2}, \\ \therefore \omega_1 &= .9091; \\ \omega_2 &= \frac{1^2}{10(1)+1^2}, \\ \therefore \omega_2 &= .0909 \end{aligned} \tag{2.36}$$

4) Using the condensation weights (2.36), condense the positively powered terms into a single term.

$$\begin{aligned} &\left[\frac{10Q}{.9091}\right]^{.9091} \times \left[\frac{Q^2}{.0909}\right]^{.0909} \\ &= 11Q^{1.0909} \end{aligned} \tag{2.37}$$

5) Using the condensed positively powered terms, and condensed negatively powered terms, state the simplified objective function.

$$z_{simplified} = 11Q^{1.0909} + 1000Q^{-1} \tag{2.38}$$

6) Use Woolsey's rule II to solve for the deltas in the simplified objective function.

The exponent matrix is

$$\begin{aligned} \delta_1 + \delta_2 &= 1 \\ 1.091\delta_1 - \delta_2 &= 0 \end{aligned} \tag{2.39}$$

Solving the system of equations gives δ_1 =.4782, and δ_2 =.5218.

7) Use the deltas calculated in step six, x_{bar}, and the harmonic mean approximation to calculate a value for the cost.

$$\begin{aligned} Cost &= \left[\frac{.4782^2}{11(1)^{1.0909}} + \frac{.5218^2}{1000(1)^{-1}} \right]^{-1} \\ &= 47.5 \end{aligned} \tag{2.40}$$

8) Use the value calculated for the cost in step seven, the appropriate delta calculated in step six, and Woolsey's rule III to calculate a new value for x.

$$\begin{aligned} &\frac{11Q^{1.0909}}{.4782} = 47.5, \\ &\therefore Q = 1.89 \end{aligned} \tag{2.41}$$

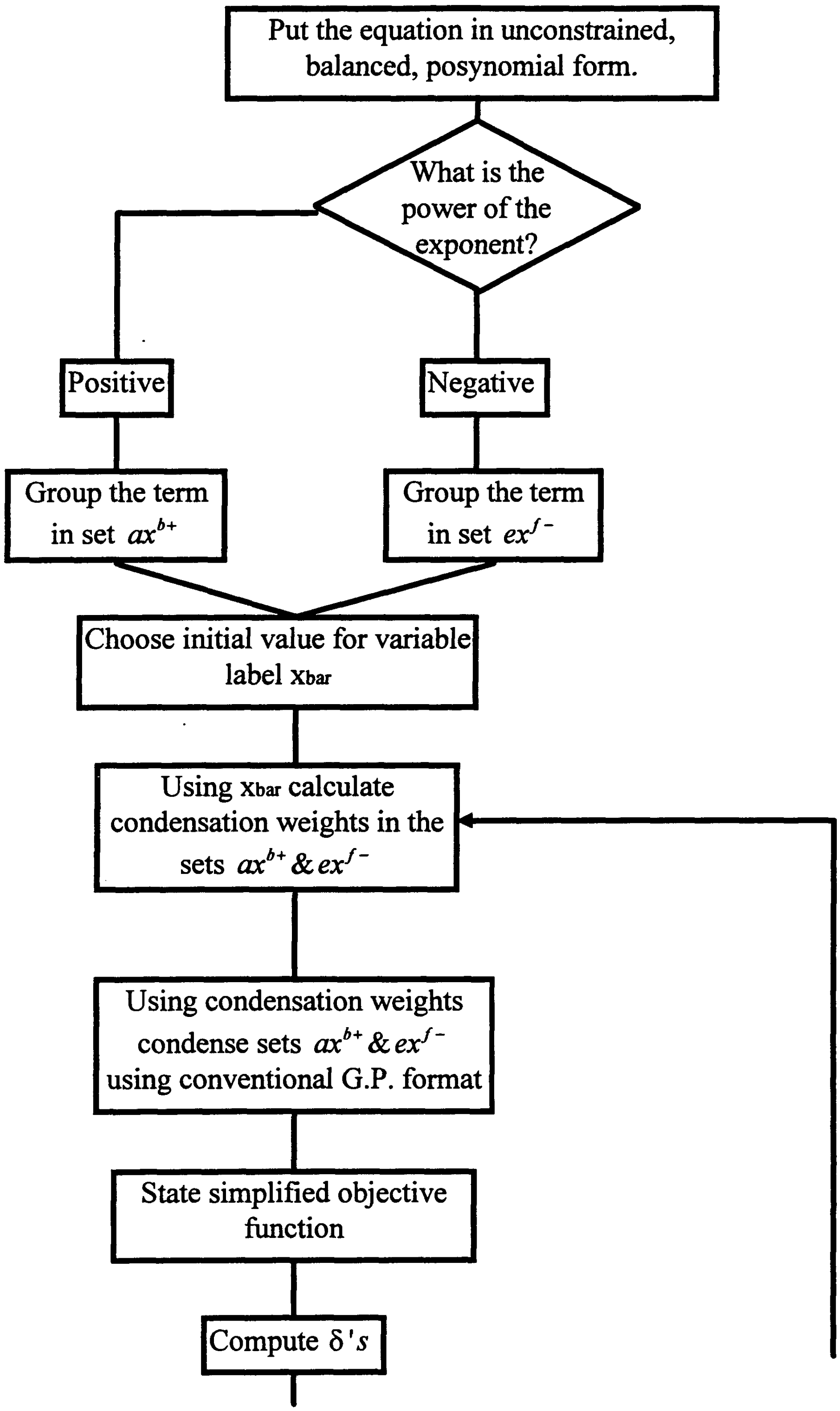
9) Compare the difference between x_{bar} and x_{new}. If the difference is negligible, substitute the value of x_{new} into the original objective function. If the difference is not, label $x_{bar} = x_{new}$, and return to step three. For this example, using epsilon = .000001, this process repeats itself for 8 iterations, until it converges at

Cost* = 261.07

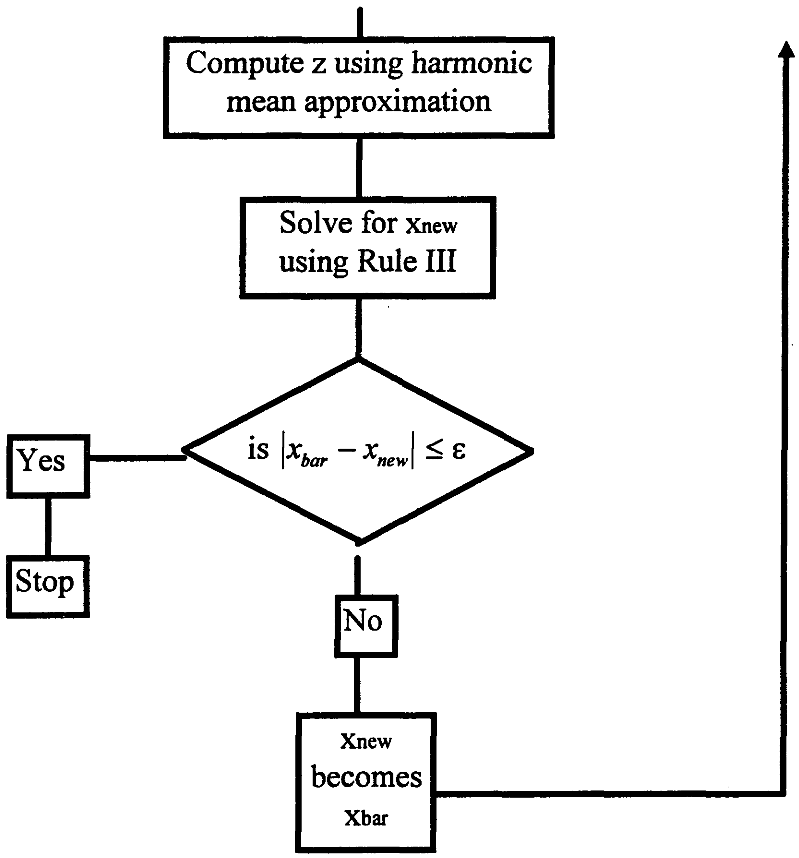
$$Q^*=6.57$$

The flowchart for harmonic programming algorithm #2 is shown on the following page.

Harmonic Programming Algorithm #2 Flowchart



Harmonic Programming Algorithm #2 Flowchart (continued):



2.4 Harmonic Programming Algorithm #3: An Algorithm to Solve Unconstrained, Multiple Degree of Difficulty, Multiple Variable, Posynomial Optimization Problems in the Form of Equation (2.2)

Algorithm #3 combines the first two algorithms to solve unconstrained, multiple degree of difficulty, multiple variable, posynomial optimization problems in the form of equation (2.2). The approach is as follows

1) Pick starting values for each variable.

2) Using the starting values, or last value calculated for each variable, treat all variables but one (x_j) as constants. Restate the problem.

3) If the simplified problem is zero degree of difficulty:

a. Solve the simplified problem using harmonic programming algorithm #1, and back out a new value for x_j.

b. If after consecutive iterations, the value for each variable does not change significantly, then stop; if not, set all but the next variable in the problem as constants, and return to step 2.

If the simplified problem is not zero degree of difficulty:

a. Solve the simplified problem using harmonic programming algorithm #2, and back out a new value for x_j.

b. If after consecutive iterations, the value for each variable does not change significantly then stop; if not, set all but the next variable in the problem as constants, and return to step 2.

Like algorithm #2, this algorithm uses a technique similar to that used by Ratliff in MULTICON, with the exception that the harmonic mean approximation is used in place of the geometric mean approximation. A step by step example follows.

The modification of the pipeline design problem as reported by Woolsey (1993), is one which can be solved using harmonic programming algorithm #3. The problem is

$$\text{Minimize:} \quad Cost = .225D^{1.47} + .475N^{-1}D^{.337} + .668N + .785D^{-.47}, \tag{2.42}$$

where D is the diameter of the pipe, and N is the number of pumping stations.

1) Choose starting values for D and N. Label the values D_{old} and N_{old}.

$D_{old} = 1$

$N_{old} = 1$

2) Treat all variables but one as constants. State the simplified problem.

For example: on the first iteration, D will be treated as a variable and N as a constant. Using the starting value for N the simplified objective function is

$$z_{simplified} = .225D^{1.47} + .475D^{.337} + .785D^{-.47}. \tag{2.43}$$

Note that the constant .668 is not used in the simplified objective function.

3) Is the simplified objective function zero degree of difficulty? If yes, then solve for a new value of D using one iteration of harmonic programming algorithm #1. If no, then solve for a new value of D using one iteration of harmonic programming algorithm #2.

For this example, it is one degree of difficulty so it is necessary to use harmonic programming algorithm #2.

a. The problem is first condensed using the techniques described previously in this thesis, which yields the following zero degree of difficulty problem

$$z_{simplified} = .7D^{.7012} + .785D^{-.47}. \tag{2.44}$$

b. Using Woolsey's rule II yields: $\delta_1 = .4013$ and $\delta_2 = .5987$ for the simplified objective function.

c. Using the deltas from b., and the starting value for D, a value for $z_{simplified}$ is calculated using the harmonic mean approximation.

$$\begin{aligned} z_{simplified} &= \left[\frac{.4013^2}{.7(1)^{.7012}} + \frac{.5987^2}{.785(1)^{-.47}} \right]^{-1} \\ &= 1.456 \end{aligned} \tag{2.45}$$

d. Using the value for $z_{simplified}$ calculated in c., the appropriate delta, and Woolsey's rule III, a new value for D is calculated.

$$\begin{aligned} &\frac{.7D^{.7012}}{.4013} = 1.456, \\ &\therefore D_{new} = .7731 \end{aligned} \tag{2.46}$$

4) Set the first variable D as a constant, and use N as a variable. State the simplified objective function. Using the new value for D (.7731), the simplified objective function is

$$z_{simplified} = .436N^{-1} + .668N. \tag{2.47}$$

Once again, note that all constants are dropped from the simplified objective function.

5) Is the simplified objective function zero degree of difficulty? If yes, then solve for a new value of N using one iteration of harmonic programming algorithm #1. If no, then solve for a new value of N using one iteration of harmonic programming algorithm #2. The new simplified objective function (2.47) is zero degree of difficulty; therefore harmonic programming algorithm #1 will be used.

a. Using Woolsey's rule II yields: $\delta_1 = .5$ and $\delta_2 = .5$ for the simplified objective function.

b. Using the deltas from a., and the starting value for N, a value for $z_{simplified}$ is calculated using the harmonic mean approximation.

$$z_{simplified} = \left[\frac{.5^2}{..436(1)^{-1}} + \frac{.5^2}{.668(1)} \right]^{-1} \\ = 1.055 \tag{2.48}$$

d. Using the value for $z_{simplified}$ calculated in b., the appropriate delta, and Woolsey's rule III, a new value for N is calculated.

$$\frac{.668N}{.5} = 1.055, \\ \therefore N_{new} = .7899 \tag{2.49}$$

6) Compare the difference between D_{old} and D_{new}, and N_{old} and N_{new}. If the difference is negligible, plug the new values for each variable into the objective function and stop. If

the difference is not, label $D_{old} = D_{new}$, and $N_{old} = N_{new}$, and return to step 2. For this example, using epsilon = .000001, this process repeats itself for 10 iterations until it converges at:

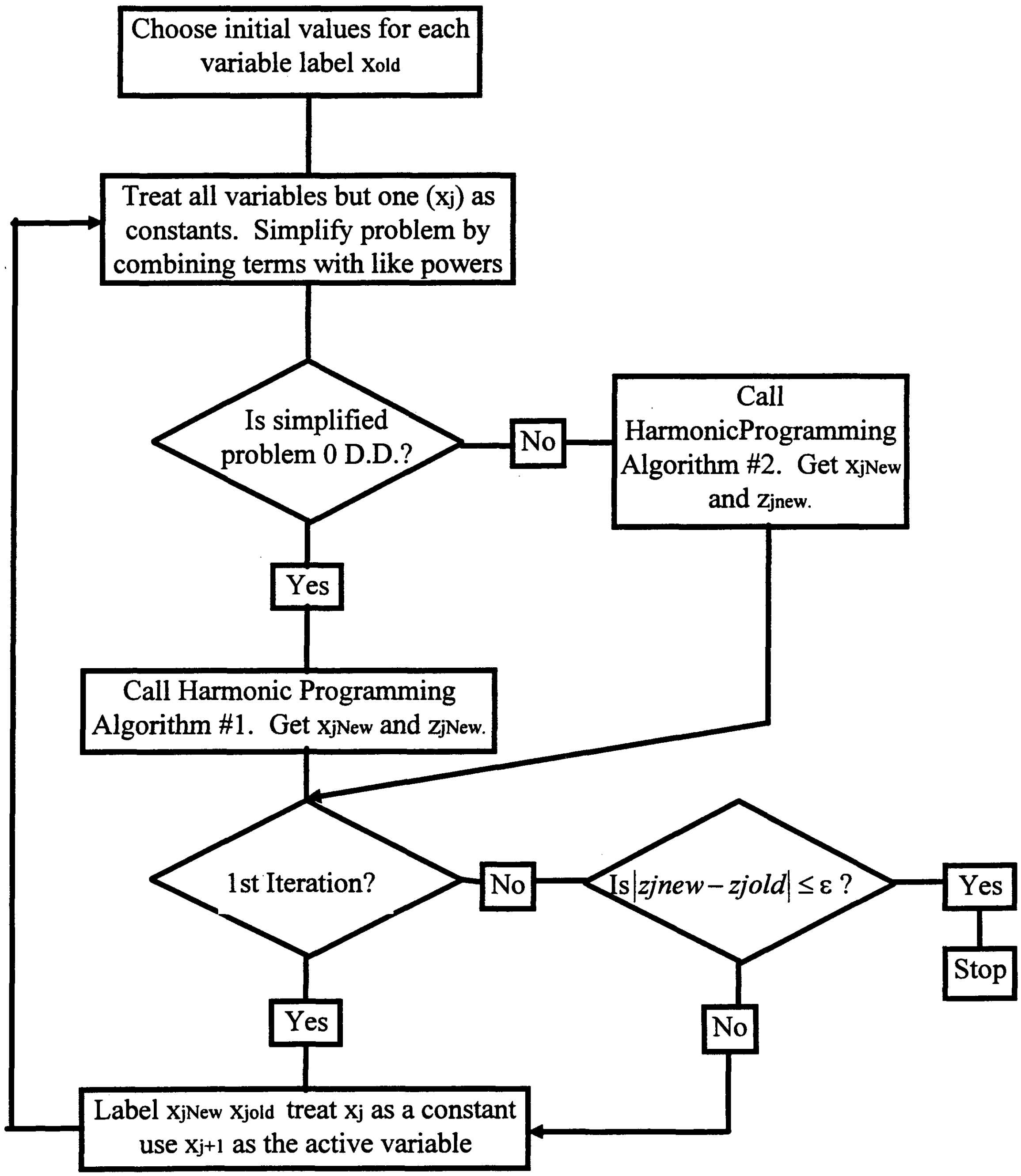
Cost = 2.1188,

D = .7842;

N = .8094.

The flowchart for harmonic programming algorithm #3 is shown on the following page.

Harmonic Programming Algorithm #3 Flowchart

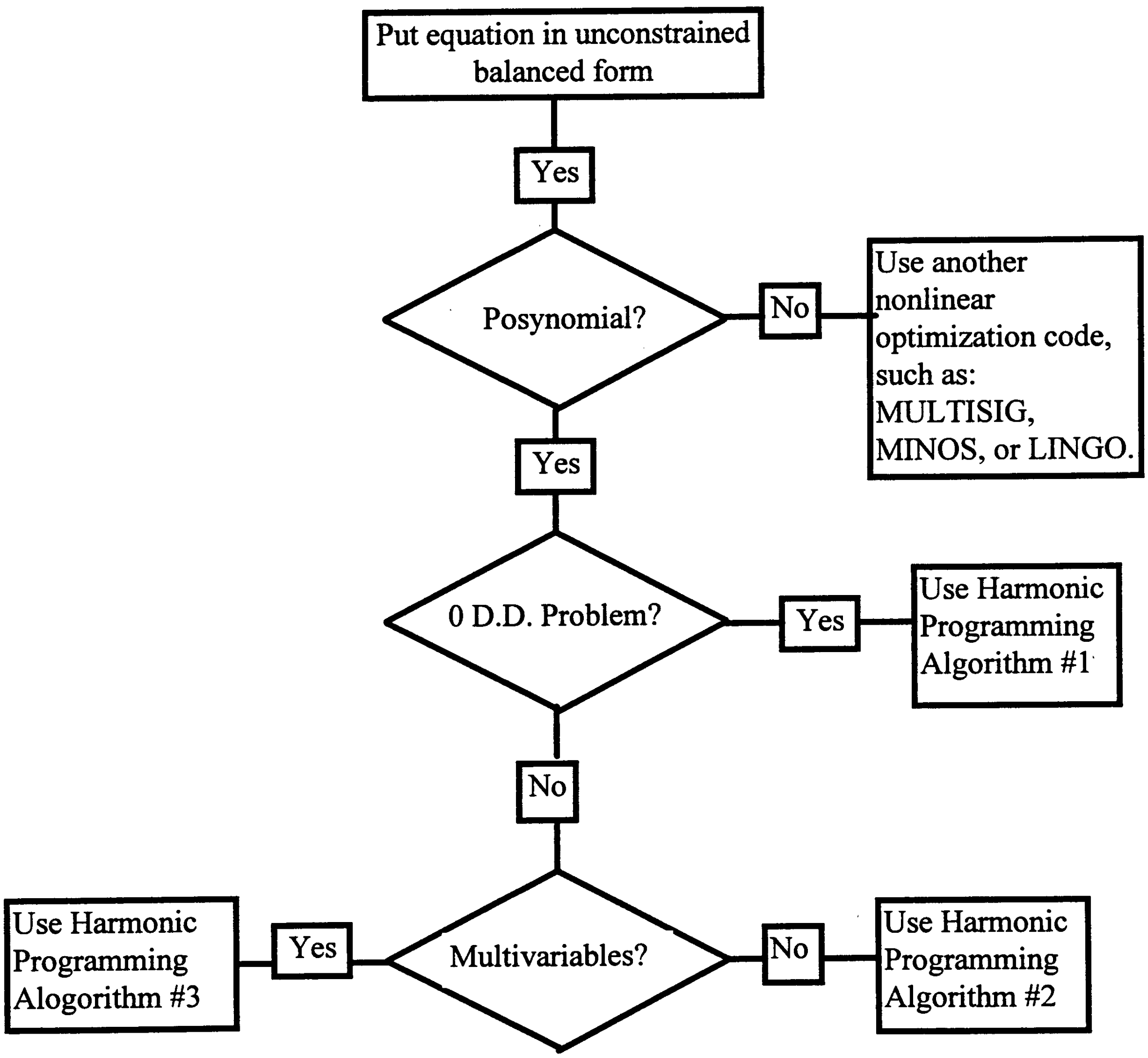


2.5 The General Algorithm for Harmonic Programming

In the previous three sections, harmonic programming algorithms 1, 2 and 3 were discussed. Combining these three algorithms produces a single algorithm which will solve unconstrained, posynomial, zero or multiple degree of difficulty, as well as single or multiple variable optimization problems in the form of equation (2.1).

The flowchart for this algorithm is shown on the following page. Using the general algorithm for harmonic programming, a computer program was written in FORTRAN for use on personal computers. The program listing is found in appendix B and the program is further discussed in the next chapter. A sample computer run for harmonic programming is found in appendix C.

General Algorithm Flowchart



Chapter 3

ALGORITHM COMPARISON

3.1 The Test Algorithms

Once the harmonic programming algorithm was coded, the next step was to compare it to the software for three established algorithms. The algorithms used for this comparison were MULTICON, MINOS, and LINGO. A brief description of each follows.

3.2 MULTICON (Ratliff, 1986)

As discussed in chapter 1, MULTICON is the program for Ratliff's algorithm. MULTICON is a generalized algorithm which solves unconstrained, balanced, multi-variable, posynomial problems using geometric programming. In Jackson's Ph. D. Thesis (1994) he found that MULTICON produced excellent results for unconstrained posynomial problems. Specifically, MULTICON will converge to the one and only local minimum which also is the global minimum (Jackson, 1994 p. 87). Since MULTICON is the closest algorithm to harmonic programming, it was chosen as one of the test programs for the comparison. A version of MULTICON written in BASIC was used for the test.

3.3 MINOS (Murtagh and Saunders, 1983)

MINOS is a software package which is generally accepted by the mathematical programming community as the baseline by which new algorithms are tested. Although it is slower than many of the newer algorithms, it has proven over time to be reliable. As described by Jackson (1994), a new algorithm typically must demonstrate that it is at least superior in speed and equal in reliability to MINOS before the nonlinear programming community is willing to exert any effort on it. The best current codes are generally 3-5 times faster than MINOS.

For the types of problems (nonlinear, unconstrained) which are solved by Harmonic Programming, MINOS employs a reduced gradient method with quasi-Newton line searches. This line search, on most problems, will provide superlinear convergence (Jackson 1994 pp. 38-39).

3.4 LINGO (Liebman, *et al* 1986)

The primary algorithm used by LINGO is a version of the generalized reduced gradient method called GRG2. GRG2 uses a reduced gradient method like MINOS, but rather than employing a single line search method, LINGO chooses from a menu of line search techniques. GRG2 will then choose the technique that it has heuristically determined to produce the quickest, most efficient results. Since GRG2 is not confined to a single line search technique, it is generally several times faster than MINOS.

Through continuous testing and improvements, LINGO remains competitive with comparable software packages (Jackson 1994 pp. 40-43).

3.5 The Test Set

The test set for the comparison consists of 23 unconstrained, posynomial optimization problems ranging from zero to five degrees of difficulty. The test set was compiled from a variety of sources, and is comprised of a majority of "real world" optimization problems. A table listing each test problem, degrees of difficulty, and reference is found on the next two pages.

3.6 Summary Table of the Problem Set

EQUATION	DD	Reference
1) $z = 78x_1 + 27x_1^{-1}x_2^{-1} + 58x_2$	0	Thome (1988) Pg. 17
2) $z = 40L^{-1}H^{-1}W^{-1} + 10LW + 20LH + 40HW$	0	Woolsey (1992) pg. 1-13 Gravel Box Design Problem
3) $z = \$316.2S^{.5} + \$34.3P + \$10^8 P^{-1}S^{-.5}$	0	Woolsey (1992) pg. 1-9 Plastic Batch Reactor
4) $z = 1.43x^{-1} + 1658.8x^{-1}s^{-1} + 47.6x^{.9}s^{.36}$	0	Woolsey (1992) pg. 1-18 Pumping Coal Slurry Problem
5) $z = 3660x + 175x^2 + 1.34x^3 + 50{,}000x^{-1}$	2	Proposed by Neghabat and Stark (1972), reported by Wilde (1978) Cofferdam Problem
6) $z = 40H^{-1}L^{-1}W^{-1} + 10LW + 20HL + 40HW$ $+10L$	1	Duffin, Peterson and Zener (1967) Gravel Sled Problem
7) $z = 4x_1x_2 + 3x_1^{-2} + 2x_1^2x_2^{-1}$	0	Woolsey G.P. Handout
8) $z = 10Q + 1000Q^{-1} + Q^2$	1	Woolsey (1992) pg. 2-4 EOQ Model for Nuclear Medicine
9) $z = .225D^{1.47} + .475N^{-1}D^{.337} + .668N + .785D^{-.47}$	1	Woolsey G.P. Exam 93, Prob. #5 Pipeline Design Problem
10) $z = 62 \cdot 10^7 s^{-3} + 25 \cdot 10^{-4} s^2 t + 96 \cdot 10^{-4} s^2$ $+35 \cdot 10^4 s^{-1}(t + 1.2)^{-1}$	2	Wilde (1978) Fruit Van Design Problem
11) $z = 10Q^{1.2}P^{-1} + 600Q^{-1} + 10^{-6}P$	0	Schweyer (1955) Batchsize Problem
12) $z = C_{23}A + C_{19}G + C_{20}G^{2.8}N^{-1.8} + C_{21}A^{-1}$ $+C_{22}A^{-1}G^{-.8}N^{.8} + C_{23}G^{-1}$ (assume all constants = 1)	2	Sherwood (1970) Ammonia Refrigerator Problem

EQUATION	DD	Reference
13) $z = .968\cdot 10^6 D^{1.63} + 2.88\cdot 10^6 D^{1.63} N^{-1}$ $+.31\cdot 10^6 D^{-4.87} + .217\cdot 10^6 N$	1	Woolsey (1992) pg. 3-5 Pipeline Pumping Station Problem #1
14) $z = 10^6 D^{1.8} + 3\cdot 10^6 D^{1.8} N^{-1} + 3\cdot 10^6 D^{-4.87}$ $+.15\cdot 10^6 N$	1	Woolsey(1992) pg. 3-6 Pipeline Pumping Station Problem #2
15) $z = 1000x + 4\cdot 10^9 x^{-1} y^{-1} + 2.5\cdot 10^5 y + 9000xy$	1	Beightler and Phillips (1976) Chemical Plant Problem
16) $z = 70.0035HL + 2333.33L^{-1} + 3333.33H^{-1}$ $+8333.33H^{-1}L^{-1}$	1	Taylor (1986) Mining Problem
17) $z = 5000T^{.5} + 25000T^{-.5}$	0	Woolsey (1975) Optimum Bitcycle Selection Problem
18) $z = 30s + 100s^{-1} + 40$	0	Schweyer (1955) Steampipe Insulation Problem
19) $z = 11.8609822x^{.470} + 441.1192843x^{-.146}$ $+3.218347592x^{.648} + 1467706.463x^{.568}$ $+1040x + 0.077708883x^{.736} + 23.68803092x^{-.229}$	5	Ratliff (1986) Space Shuttle Design Problem
20) $z = .1\left[12 + x^2 + \frac{1+y^2}{x^2} + \frac{x^2y^2+100}{(xy)^4}\right]$	2	Ravindran *et al* (1983) Gear Train Inertia Problem
21) $z = 5xy + 7x + 8y + 4x^{-2} + 8y^{-2}$	2	Wessels (1989) Wessels Problem 1
22) $z = 60x^{-3}y^{-2} + 50x^3y + 20x^{-3}y^3$	0	Reklaitis *et al* Problem pg. 499 (1983)
23) $z = (xy)^{-1} + x^{.5} + y^{.75}$	0	Reklaitis *et al* Problem pg. 531 (1983)

3.7 The Comparison

Since the software for each of the algorithms used in this comparison, are written in different computer languages, and by different programmers, this comparison is not a truly fair one. Each of the algorithms is constrained by how quickly its respective language can process the code. In order to make the comparison as impartial as possible, the software for each of the four algorithms was loaded on the same computer. The computer was a 386 EVEREX PC, with 8 megabytes of RAM. Each test problem was then solved using each of the algorithms. A starting value of 1 was given to each of the variables for every problem. This value was used since it was an unbiased number, and because previous algorithm comparisons (Dolan, Jackson) had also used it as an initial value. The number of iterations, running time, and solutions were recorded. A summary table of these results is given on the following three pages, and a complete listing of these test results can be found in appendix A.

3.8 Summary Table of the Comparison Results

EQUATION	Optimal Solution	Solution Found & Running Time			
		H. P.	MULTICON	MINOS	LINGO
1) $z = 78x_1 + 27x_1^{-1}x_2^{-1} + 58x_2$	z* = 148.85 x1* = .6361 x2* = .8555	z* = 148.85 x1* = .6361 x2* = .8555 time:< 1 sec	z* = 148.85 x1* = .6361 x2* = .8555 time: 8 sec	z* = 148.85 x1* = .636 x2* =.855 time: 8 sec	z* = 148.85 x1* = .6367 x2* = .8556 time < 1 sec
2) $z = 40L^{-1}H^{-1}W^{-1} + 10LW + 20LH + 40HW$	z* = 100 L* = 2.00 H* = .500 W* = 1.00	z* = 100 L* = 1.9999 H* = .5000 W* = .9999 time:< 1 sec	z* = 100 L* = 2.0006 H* = .4999 W* = .9998 time: 24 sec	z* =100 L* = 2.000 H* = .500 W* = 1.000 time: 11 sec	z* = 100 L* = 1.9968 H* = .5021 W* = 1.000 time: 1 sec
3) $z = \$316.2S^{.5} + \$34.3P + \$10^8 P^{-1}S^{-.5}$	z*= $30,822 S*=1055.80 P* = 299.54	z* = 30,822 S*=1055.80 P* = 299.54 time:< 1 sec	z* = 30,822 S*=1055.89 P* = 299.54 time: 8 sec	z* = 30,822 S*=1055.80 P* = 299.54 time: 8 sec	z* = 30,822 S*=1055.76 P* = 299.54 time: 2 sec
4) $z = 1.43x^{-1} + 1658.8x^{-1}s^{-1} + 47.6x^{.9}s^{.36}$	z* = 94.60 x* = .0532 s* = 1740	z* = 94.63 x* = .0532 s* =1739.99 time:< 1 sec	z* = 94.63 x* = .0532 s* =1739.71 time:21 sec	z* = 94.63 x* = .053 s* =1740.00 time: 9 sec	z* = 131.45 x* = .2071 s* = 144.63 time:7 sec
5) $z = 3660x + 175x^2 + 1.34x^3 + 50{,}000x^{-1}$	z* = 29,172 x* = 3.218	z* = 29,172 x* = 3.218 time:< 1 sec	z* = 29,172 x* = 3.218 time: 1 sec	z* = 29,172 x* = 3.218 time: 8 sec	z* = 29,172 x* = 3.218 time: 1 sec
6) $z = 40H^{-1}L^{-1}W^{-1} + 10LW + 20HL + 40HW$ $+10L$	z* =115.72 H* = .5962 L* = 1.2942 W = 1.1884	z* = 115.72 H* = .5949 L* = 1.2930 W*=1 .1899 time:< 1 sec	z* = 115.72 H* = .5949 L* = 1.2930 W*= 1.1899 time: 25 sec	z* = 115.72 H* = .595 L* = 1.293 W* = 1.190 time: 9 sec	z* = 115.72 H* = .5950 L* = 1.2929 W*= 1.1899 time: 3 sec
7) $z = 4x_1x_2 + 3x_1^{-2} + 2x_1^2x_2^{-1}$	z* = 8.533 x1* = .906 x2* = .673	z* = 8.533 x1* = .9057 x2* = .6730 time:< 1 sec	z* = 8.533 x1* = .9057 x2* = .6730 time: 4 sec	z* = 8.533 x1* = .906 x2* =.673 time: 8 sec	z* = 8.533 x1* = .9057 x2* = .6730 time: 1 sec
8) $z = 10Q + 10000Q^{-1} + Q^2$	z* = 261.07 Q* = 6.57	z* = 261.07 Q* = 6.57 time:< 1 sec	z* = 261.07 Q* = 6.57 time: 1 sec	z* = 261.07 Q* = 6.57 time: 7 sec	z* = 261.07 Q* = 6.57 time:< 1 sec

EQUATION	Optimal Solution	Solution Found & Running Time H.P.	MULTICON	MINOS	LINGO
9) $z = .225D^{1.47} + .475N^{-1}D^{337} + .668N + .785D^{-.47}$	z* = 2.1188 D* = .7848 N* = .8099	z* = 2.1188 D* = .7842 N* = .8094 time: < 1 sec	z* = 2.1188 D* = .7842 N* = .8094 time: 7 sec	z* = 2.1188 D* = .784 N* = .809 time: 8 sec	z* = 2.1188 D* = .7847 N* = .8097 time: 1 sec
10) $z = 62 \cdot 10^7 s^{-3} + 25 \cdot 10^{-4} s^2 t + 96 \cdot 10^{-4} s^2$ $+35 \cdot 10^4 s^{-1}(t+1.2)^{-1}$	z* = 1054.42 s* = 143.68 t* = 5.67	z* = 1054.42 s* = 143.68 t* = 5.67 time: < 1 sec	z* = 1072.97 s* = 128.05 t* = 6.96 time: 8 sec	z* = 1054.42 s* = 143.68 t* = 5.67 time: 9 sec	z* = 1054.42 s* = 143.68 t* = 5.67 time: 2 sec
11) $z = 10Q^{1.2}P^{-1} + 600Q^{-1} + 10^{-6}P$	z* = .9033 Q* = 1809.78 P* = 279,687	z* = .9011 Q* = 1775 P* = 281,587 time: < 1 sec	z* = .9011 Q* = 1775 P* = 281,587 time: 9 sec	z* = .9011 Q* = 1776 P* = 281,600 time: 8 sec	z* = 1.3652 Q* = 835 P* = 54,113 time: 12 sec
12) $z = C_{23}A + C_{19}G + C_{20}G^{2.8}N^{-1.8} + C_{21}A^{-1}$ $+C_{22}A^{-1}G^{-.8}N^{.8} + C_{23}G^{-1}$ (assume all constants = 1)	z* = 5.5222 A* = 1.5426 G* = .8202 N* = 1.2264	z* = 5.5222 A* = 1.5428 G* = .8207 N* = 1.2277 time: < 1 sec	z* = 5.5222 A* = 1.5420 G* = .8168 N* = 1.2194 time: 29 sec	z* = 5.5222 A* = 1.543 G* = .821 N* = 1.228 time: 8 sec	z* = 5.5232 A* = 1.5586 G* = .8409 N* = 1.2944 time: 1 sec
13) $z = .968 \cdot 10^6 D^{1.63} + 2.88 \cdot 10^6 D^{1.63} N^{-1}$ $+.31 \cdot 10^6 D^{-4.87} + .217 \cdot 10^6 N$	z* = 2,784,080 D* = .9006 N* = 3.3451	z* = 2,784,080 D* = .9006 N* = 3.3451 time: < 1 sec	z* = 2,784,080 D* = .9006 N* = 3.3451 time: 5 sec	z* = 2,784,080 D* = .901 N* = 3.345 time: 8 sec	z* = 2,784,080 D* = .9006 N* = 3.3451 time: 5 sec
14) $z = 10^6 D^{1.8} + 3 \cdot 10^6 D^{1.8} N^{-1} + 3 \cdot 10^6 D^{-4.87}$ $+.15 \cdot 10^6 N$	z* = 4,136,385 D* = 1.2835 N* = 5.5985	z* = 4,136,385 D* = 1.2835 N* = 5.5985 time: < 1 sec	z* = 4,136,385 D* = 1.2835 N* = 5.5985 time: 5 sec	z* = 4,136,385 D* = 1.284 N* = 5.599 time: 9 sec	z* = 4,136,406 D* = 1.2839 N* = 5.6277 time: 2 sec
15) $z = 1000x + 4 \cdot 10^9 x^{-1} y^{-1} + 2.5 \cdot 10^5 y + 9000xy$	z*=12,809,668 x* = 401.565 y* = 1.60557	z*=12,809,668 x* = 401.48 y* = 1.6059 time: 3 sec	z*=12,809,668 x* = 401.48 y* = 1.6059 time: 104 sec	z*=12,809,668 x* = 401.48 y* = 1.606 time: 10 sec	z*=12,819,650 x* = 344.24 y* = 1.8819 time: 4 sec

EQUATION	Optimal Solution	Solution Found & Running Time			
		H.P.	MULTICON	MINOS	LINGO
16) $z = 70.0035HL + 2333.33L^{-1} + 3333.33H^{-1} + 8333.33H^{-1}L^{-1}$	z* = 3032 H* = 4.899 L* = 3.429	z* = 3032.92 H* = 4.899 L* = 3.430 time: < 1 sec	z* = 3032.92 H* = 4.899 L* = 3.430 time: 16 sec	z* = 3032.92 H* = 4.899 L* = 3.430 time: 9 sec	z* = 3032.92 H* = 4.898 L* = 3.432 time: 1 sec
17) $z = 5000T^{.5} + 25000T^{-.5}$	z* = 22,360.67 T*= 5	z* = 22,360.68 T* =5.0 time: < 1 sec	z* = 22,360.68 T* = 5.0 time: < 1 sec	z* = 22,360.68 T* = 5.0 time: 8 sec	z* = 2,360.68 T* = 5.0 time: < 1 sec
18) $z = 30s + 100s^{-1} + 40$	z* = 149.54 s* = 1.826	z* = 149.54 s* = 1.826 time: < 1 sec	z* = 149.54 s* = 1.826 time: < 1 sec	z* = 149.54 s* = 1.826 time: 9 sec	z* = 149.54 s* = 1.826 time: < 1 sec
19) $z = 11.8609822x^{.470} + 441.1192843x^{-.146} + 3.218347592x^{.648} + 1467706.463x^{.568} + 1040x + 0.077708883x^{.736} + 23.68803092x^{-.229}$	z* = 4319.55 x* = .00000237	z* = 4319.55 x*= .00000237 time: < 1 sec	z* = 4319.55 x*= .00000237 time: 1 sec	z* = 4319.55 x*= .00000237 time: 10 sec	z* = 4319.55 x*=.00000237 time: 1 sec
20) $z = .1\left[12 + x^2 + \frac{1+y^2}{x^2} + \frac{x^2y^2+100}{(xy)^4}\right]$	z* = 1.74415 x* = 1.74345 y* = 2.02969	z* = 1.7442 x* = 1.7435 y* = 2.0297 time: < 1 sec	z* = 1.7461 x* = 1.6808 y* = 2.041 time: 8 sec	z* = 1.7442 x* = 1.743 y* = 2.030 time: 10 sec	z* = 1.7442 x* = 1.7435 y* = 2.0297 time: 1 sec
21) $z = 5xy + 7x + 8y + 4x^{-2} + 8y^{-2}$	z* = 31.5686 x* = .862787 y* = 1.09121	z* = 31.5686 x* = .8628 y* = 1.0912 time: < 1 sec	z* = 31.5686 x* = .8628 y* = 1.0912 time: 3 sec	z* = 31.5686 x* = .863 y* = 1.091 time: 9 sec	z* = 31.5687 x* = .8646 y* = 1.0926 time: < 1 sec
22) $z = 60x^{-3}y^{-2} + 50x^3y + 20x^{-3}y^3$	z* = 126.049 x * = 1.10114 y* = .944088	z* = 126.049 x* = 1.1011 y* = .9441 time: < 1 sec	z* = 126.049 x* = 1.1011 y* = .9441 time: 11 sec	z* = 126.049 x* = 1.101 y* = .944 time: 9 sec	z* = 126.049 x* = 1.1012 y* = .9441 time: 1 sec
23) $z = (xy)^{-1} + x^{.5} + y^{.75}$	z* = 2.88033 x*= 1.76726 y* = .851293	z* = 2.8803 x* = 1.7673 y* = .8513 time: < 1 sec	z* = 2.8803 x* = 1.7667 y* = .8515 time: 4 sec	z* = 2.8803 x* = 1.767 y* = .851 time: 9 sec	z* = 2.8805 x* = 1.7571 y* = .8701 time: 1 sec

3.9 Comparison Results & Table

The following table illustrates how harmonic programming compared to the other algorithms, subject to the caveat on page 43. An "X" in a column signifies that harmonic programming was faster or produced more accurate results (at least .0001closer to the optimal solution), than the specified program. A "--" signifies no difference. A "w" signifies that harmonic programming was slower or produced less accurate results.

	MULTICON	MULTICON	MINOS	MINOS	LINGO	LINGO
Problem #	RUN TIME	ACCURACY	RUN TIME	ACCURACY	RUN TIME	ACCURACY
1	X	--	X	--	--	X
2	X	X	X	--	X	X
3	X	X	X	--	X	X
4	X	X	X	--	X	X
5	X	--	X	--	X	--
6	X	--	X	--	X	--
7	X	--	X	--	X	--
8	X	--	X	--	--	--
9	X	--	X	--	X	--
10	X	X	X	--	X	--
11	X	--	X	--	X	X
12	X	X	X	--	X	X
13	X	--	X	--	X	--
14	X	--	X	--	X	X
15	X	--	X	--	X	X
16	X	--	X	--	X	X
17	--	--	X	--	--	--
18	--	--	X	--	--	--
19	X	--	X	--	X	--
20	X	X	X	--	X	--
21	X	--	X	--	--	X
22	X	--	X	--	X	--
23	X	X	X	--	X	X

The comparison showed, subject to the caveat on page 43, that for every test problem, harmonic programming produced as good, or (in most cases) better results in running time and accuracy than the other three algorithms. Specifically, harmonic programming was more accurate and faster than MULTICON and LINGO, and as accurate and faster than MINOS.

Chapter 4

CONCLUSIONS & SUGGESTIONS FOR FURTHER STUDY

4.1 Conclusion

The harmonic programming algorithm shows that, like the geometric mean, the harmonic mean can be used in mathematical programming. Most significantly, not only did the algorithm work, but for every test problem, subject to the caveat on page 43, harmonic programming produced as good or (in most cases) better results in running time and accuracy than MULTICON, MINOS, and LINGO. Also important is the fact that, since harmonic programming is a whole new field in mathematical programming, it opens many doors for further research.

4.2 Limitations

Although harmonic programming has been shown to be successful for solving unconstrained posynomial functions, its greatest limitations are its inability to solve constrained posynomial functions, and signomial problems. The author tried, unsuccessfully, to hand calculate a few of these problems, but this area was not researched extensively.

4.3 Areas for Further Research

Signomials have always been a topic of many studies. Most recently, William T. Dolan developed an algorithm (MULTISIG) to solve unconstrained signomials. He discovered that, given a signomial function, one could bring all negative terms to the left hand side, condense it, divide back through and solve the new posynomial with Ratliff's method. Since harmonic programming has been shown to be as accurate as and faster than MULTICON, it is logical to assume that harmonic programming could replace Ratliff's method in MULTISIG. The resulting algorithm would probably be faster than MULTISIG.

Another interesting point about signomials involves Woolsey's rule II exponent matrix. In geometric programming, if a zero degree of difficulty signomial is encountered, it is possible to solve the problem using slight modifications to Woolsey's rules II and III. Using the same modification to the rule II matrix, I was able to solve by hand a couple of zero degree of difficulty signomial problems using harmonic programming. One of these problems was

$$\text{Minimize: } z = 30x^3 - 10x. \tag{4.1}$$

The signomial rule II matrix is built in the same way as posynomials, with two exceptions. The first exception is that the right hand side of the first row is represented as σ. Sigma will be either 1 or -1, depending on which value will produce positive deltas when the system of equations is solved. The second exception is that each delta in the matrix is also multiplied by the power of its corresponding coefficient. For the problem above, the exponent matrix would be as follows

$$\begin{aligned} \delta_1 - \delta_2 &= \sigma \\ 3\delta_1 - \delta_2 &= 0. \end{aligned} \tag{4.2}$$

For this example, using $\sigma = -1$, and solving the system of equations yields: $\delta_1 = .5$, and $\delta_2 = 1.5$. Using these deltas, and harmonic programming algorithm #1, the optimal value of x was calculated to be .333, and the optimal value for z was -2.22. These values match the optimal values which result when using calculus. Since this technique works for some signomials, another area for research would be to determine for what kinds of signomials this method will work.

A third area for research would be to expand the harmonic programming algorithm so that it would be able to solve constrained problems.

A fourth area for research would be to see whether the logarithmic mean could be used to develop an algorithm similar to those for geometric and harmonic programming. We would conjecture that a hybrid algorithm using the geometric, harmonic, and logarithmic mean might well be worth investigating, in a manner similar to that of this thesis.

A final suggested topic would be to prove whether or not the squared deltas in the approximation cause harmonic programming to converge faster than geometric programming.

REFERENCES CITED

Beightler, C. and D. Phillipps. 1976. Applied Geometric Programming. New York: John Wiley and Sons.

Beyer, W.H. 1991. CRC Standard Mathematical Tables and Formulae. Boca Raton, Florida: CRC Press.

Dolan, W.T. 1995. "A Geometric Programming Algorithm for Solving a Class of Unconstrained Signomials." Master's Thesis, Colorado School of Mines.

Duffin, R.J., E.L. Peterson, and C. Zener. 1967. Geometric Programming. New York: Wiley and Sons.

Duffin, R.J., E.L. Peterson, and C. Zener. 1972. "Reversed Geometric Programs Treated by Harmonic Means." Indiana University Mathematics Journal vol. 22: 531-550.

Jackson, J.A. 1994. "A Mathematical Experiment in Geometric Programming." Ph. D. Thesis, Colorado School of Mines.

Ratliff, R.M. 1986. "A Generalized Condensation Algorithm for the Solution of Unconstrained Balanced Posynomial Problems Using Geometric Programming." Master's Thesis, Colorado School of Mines.

Reklaitis, G. V., A. Ravindran, and K. M. Ragsdell. 1983. Engineering Optimization. New York: Wiley and Sons.

Schweyer, H. 1955. Process Engineering Economics. New York: McGraw-Hill.

Sherwood, T. K. 1970. A Course in Process Design. New York: John Wiley and Sons.

Taylor, D. 1986. An unpublished report on a shrink stop mining process. University of Nevada at Reno.

Thome, J. 1988. "An Algorithm for Solving a Class of Nonlinear Unconstrained Signomial Economic Models Using the Greening Technique." Ph. D. Thesis, Colorado School of Mines.

Wessels, G. J. 1989. "A Geometric Programming Algorithm for Solving a Class of Nonlinear, Signomial Optimization Problems." Ph. D. Thesis, Colorado School of Mines.

Wilde, D. J. 1978. Globally Optimal Design. New York: Wiley and Sons.

Woolsey, R. E. D. 1992 Engineering Design Optimized with Geometric Programming. Vol. 1 in the Useful Engineering Series.

Woolsey, R. E. D. And H. S. Swanson. 1975. Operations Research for Immediate Application: A Quick and Dirty Manual. New York: Harper and Row.

GLOSSARY OF TERMS (RATLIFF 1986)

Balanced: at least one positive and one negative exponent must appear for each variable in the problem.

Condensation: a method used to reduce the degree of difficulty of a problem. It is an iterative procedure which employs the geometric inequality a second time while solving the original problem.

Constraint: a relationship which defines some bounds to the possible values of the variables in the problem.

Convergence Condition: the maximum change in the value of the function (or the values of the variables) through successive iterations which is allowed for convergence to have occurred.

Degree of Difficulty: an indication of how difficult it is to solve the original problem; progressively higher numbered problems become much more difficult to solve. The degree of difficulty is defined to be the number of terms, minus the number of variables, minus one.

Harmonic Programming: an algorithm similar to geometric programming, which uses the arithmetic-geometric-harmonic mean inequality to solve unconstrained balanced posynomials.

Iterations: a measurement of the number of solutions which are evaluated until one is found which satisfies the convergence condition.

Nonlinear: an equation in which the variables may appear with powers other than one.

Posynomial: similar to a polynomial; the coefficients must be positive and the exponents on each variable are real constants.

Term: any part of the equation set apart from the rest by a +, -, or inequality as well as having both a constant and a variable part to it.

Unconstrained: no restrictions exist on the variables or the objective function other than they be positive, real values.

<u>Weight</u>: the fraction of the total value of an equation contributed by a particular term

Appendix A

Test Problems

1) THOME PROBLEM 1 (Thome 1988)

Minimize: $z = 78x_1 + 27x_1^{-1}x_2^{-1} + 58x_2$

Optimal solution: $z = 148.85$
$x_1 = .6361$
$x_2 = .8555$

Initial values: $x_1 = 1$
$x_2 = 1$
$\gamma = .000001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	5	< 1 sec	$x_1 = .6361$ $x_2 = .8555$	148.85
MULTICON	38	8 sec	$x_1 = .6361$ $x_2 = .8555$	148.85
MINOS	6	8 sec	$x_1 = .636$ $x_2 = .855$	148.85
LINGO	4	< 1 sec	$x_1 = .6367$ $x_2 = .8556$	148.85

2) THE GRAVEL BOX PROBLEM (Woolsey 1992)

Minimize: $z = 40L^{-1}H^{-1}W^{-1} + 10LW + 20LH + 40HW$

Optimal solution: z = 100
L = 2.00
H = 0.50
W = 1.00

Initial values: L = 1
H = 1
W = 1
$\gamma = .00001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	25	< 1 sec	L = 1.9999 H = 0.5000 W = 0.9999	100
MULTICON	102	24 sec	L = 2.0006 H = 0.4999 W = 0.9998	100
MINOS	11	11 sec	L = 2.000 H = 0.500 W = 1.000	100
LINGO	12	1 sec	L = 1.9968 H = 0.5021 W = 1.000	100

3) PLASTIC BATCH REACTOR PROBLEM (Woolsey 1992)

Minimize: $z = \$316.2S^{.5} + \$34.3P + \$10^8 P^{-1}S^{-.5}$

Optimal solution: z = 30,822
S = 1055.80
P = 299.54

Initial values: S = 1
P = 1
$\gamma = .000001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	18	< 1 sec	S = 1055.80 P = 299.54	30,822
MULTICON	53	8 sec	S = 1055.89 P = 299.54	30,822
MINOS	17	8 sec	S = 1055.80 P = 299.54	30,822
LINGO	15	2 sec	S = 1055.76 P = 299.54	30,822

4) PUMPING COAL SLURRY PROBLEM (Woolsey 1992)

Minimize: $z = 1.43x^{-1} + 1658.8x^{-1}s^{-1} + 47.6x^{.9}s^{.36}$

Optimal solution: z = 94.6
x = 0.0532
s = 1740

Initial values: x = 1
s = 1
$\gamma = .000001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	14	< 1 sec	x = 0.0532 s = 1739.99	94.63
MULTICON	132	21 sec	x = 0.0532 s = 1739.71	94.63
MINOS	22	9 sec	x = .053 s = 1740.00	94.63
LINGO	65	7 sec	x = 0.2071 s = 144.63	131.45

5) **COFFERDAM PROBLEM** (Wilde 1978)

Minimize: $z = 3660x + 175x^2 + 1.34x^3 + 50{,}000x^{-1}$

Optimal solution: z = 29,172.35
x = 3.218

Initial values: x = 1
$\gamma = .000001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	8	< 1 sec	x = 3.218	29,172.35
MULTICON	6	1 sec	x = 3.218	29,172.35
MINOS	5	8 sec	x = 3.218	29,172.35
LINGO	4	1 sec	x = 3.218	29,172.35

6) GRAVEL SLED PROBLEM (Duffin, Peterson, and Zener 1967)

Minimize: $z = 40H^{-1}L^{-1}W^{-1} + 10LW + 20HL + 40HW + 10L$

Optimal solution: z = 115.72
H = .5962
L = 1.2942
W = 1.1884

Initial values: H = 1
L = 1
W = 1
$\gamma = .000001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	26	< 1 sec	H = 0.5949 L = 1.2930 W = 1.1899	115.72
MULTICON	131	25 sec	H = 0.5949 L = 1.2930 W = 1.1899	115.72
MINOS	10	9 sec	H = 0.595 L = 1.293 W = 1.190	115.72
LINGO	27	3 sec	H = 0.5950 L = 1.2929 W = 1.1899	115.72

7) **WOOLSEY PROBLEM 1** (Woolsey Handout)

Minimize: $z = 4x_1x_2 + 3x_1^{-2} + 2x_1^2x_2^{-1}$

Optimal solution: $z = 8.533$
$x_1 = 0.906$
$x_2 = 0.673$

Initial values: $x_1 = 1$
$x_2 = 1$
$\gamma = .000001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	5	< 1 sec	$x_1 = 0.9057$ $x_2 = 0.6730$	8.533
MULTICON	25	4 sec	$x_1 = 0.9057$ $x_2 = 0.6730$	8.533
MINOS	5	8 sec	$x_1 = 0.906$ $x_2 = 0.673$	8.533
LINGO	11	1 sec	$x_1 = 0.9057$ $x_2 = 0.6730$	8.533

8) EOQ MODEL FOR NUCLEAR MEDICINE (Woolsey 1992)

Minimize: $z = 10Q + 1000Q^{-1} + Q^2$

Optimal solution: z = 261.07
Q = 6.57

Initial value: Q = 1
$\gamma = .00001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	8	< 1 sec	Q = 6.57	261.07
MULTICON	7	1 sec	Q = 6.57	261.07
MINOS	3	7 sec	Q = 6.57	261.07
LINGO	2	< 1 sec	Q = 6.57	261.07

9) PIPELINE DESIGN PROBLEM (Woolsey 1993)

Minimize: $z = .225D^{1.47} + .475N^{-1}D^{.337} + .668N + .785D^{-.47}$

Optimal solution: z = 2.1188
D = 0.7848
N = 0.8099

Initial values: D = 1
N = 1
$\gamma = .000001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	10	< 1 sec	D = 0.7842 N = 0.8094	2.1188
MULTICON	48	7 sec	D = 0.7842 N = 0.8094	2.1188
MINOS	4	8 sec	D = 0.784 N = 0.809	2.1188
LINGO	4	1 sec	D = 0.7847 N = 0.8097	2.1188

10) FRUIT VAN DESIGN PROBLEM (Wilde 1978)

Minimize: $z = 62 \cdot 10^7 s^{-3} + 25 \cdot 10^{-4} s^2 t + 96 \cdot 10^{-4} s^2 + 35 \cdot 10^4 s^{-1} (t + 1.2)^{-1}$

Let: u = t + 1.2
t = u - 1.2

The new objective function is:

Minimize: $z = 62 \cdot 10^7 s^{-3} + 25 \cdot 10^{-4} s^2 u + 35 \cdot 10^4 s^{-1} u^{-1} + .0066 s^2$

Optimal solution: z = 1054.42
s = 143.68
t = 5.67

Initial values: s = 1
t = 1
γ = .0000000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	42	< 1 sec	s = 143.68 t = 5.67	1054.42
MULTICON	45	8 sec	s = 128.05 t = 6.96	1072.97
MINOS	13	9 sec	s = 143.68 t = 5.67	1054.42
LINGO	13	2 sec	s = 143.68 t = 5.67	1054.42

11) **BATCHSIZE PROBLEM** (Schweyer 1955)

Minimize: $z = 10Q^{1.2}P^{-1} + 600Q^{-1} + 10^{-6}P$

Optimal solution: z = 0.9033
Q = 1809.78
P = 279687

Initial values: Q = 1
P = 1
γ = .0000000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	37	< 1 sec	Q = 1775 P = 281,587	.9011
MULTICON	56	9 sec	Q = 1775 P = 281,587	.9011
MINOS	19	8 sec	Q = 1776 P = 281,600	.9011
LINGO	101	12 sec	Q = 835 P = 54113	1.3652

12) AMMONIA REFRIGERATOR PROBLEM (Sherwood 1970)

Minimize: $z = C_{23}A + C_{19}G + C_{20}G^{2.8}N^{-1.8} + C_{21}A^{-1} + C_{22}A^{-1}G^{-.8}N^{.8} + C_{23}G^{-1}$

(assume all constants (C) = 1)

Optimal solution: z = 5.5222
A = 1.5426
G = .8202
N = 1.2264

Initial values: A = 1
G = 1
N = 1
γ = .000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	25	< 1 sec	A = 1.5428 G = 0.8207 N = 1.2277	5.5222
MULTICON	156	29 sec	A = 1.5420 G = 0.8168 N = 1.2194	5.5222
MINOS	7	8 sec	A = 1.543 G = 0.821 N = 1.228	5.5222
LINGO	6	1 sec	A = 1.5586 G = 0.8409 N = 1.2944	5.5232

13) PIPELINE PUMPING STATION 1 (Woolsey 1992)

Minimize: $z = .968 \cdot 10^6 D^{1.63} + 2.88 \cdot 10^6 D^{1.63} N^{-1} + .31 \cdot 10^6 D^{-4.87} + .217 \cdot 10^6 N$

Optimal solution: z = 2,784,080
D = 0.9006
N = 3.3451

Initial values: D = 1
N = 1
$\gamma = .000001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	9	< 1 sec	D = 0.9006 N = 3.3451	2,784,080
MULTICON	28	5 sec	D = 0.9006 N = 3.3451	2,784,080
MINOS	10	8 sec	D = 0.901 N = 3.345	2,784,080
LINGO	33	5 sec	D = 0.9006 N = 3.3451	2,784,080

14) **PIPELINE PUMPING STATION PROBLEM 2** (Woolsey 1992)

Minimize: $z = 10^6 D^{1.8} + 3 \cdot 10^6 D^{1.8} N^{-1} + 3 \cdot 10^6 D^{-4.87} + .15 \cdot 10^6 N$

Optimal solution: z = 4,136,385
D =1.2835
N =5.5985

Initial values: D = 1
N = 1
$\gamma = .0000000001$

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	12	< 1 sec	D = 1.2835 N = 5.5985	4,136,385
MULTICON	28	5 sec	D = 1.2835 N = 5.5985	4,136,385
MINOS	9	9 sec	D = 1.284 N = 5.599	4,136,385
LINGO	16	2 sec	D = 1.2839 N = 5.6277	4,136,406

15) **CHEMICAL PLANT PROBLEM** (Beightler and Phillips 1976)

Minimize: $z = 1000x + 4 \cdot 10^9 x^{-1} y^{-1} + 2.5 \cdot 10^5 y + 9000xy$

Optimal solution: z = 12,809,668
x = 401.565
y = 1.60557

Initial values: x = 1
y = 1
γ = .0000000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	415	3 sec	x = 401.48 y = 1.6059	12,809,668
MULTICON	632	1 min 44 sec	x = 401.48 y = 1.6059	12,809,668
MINOS	21	10 sec	x = 401.48 y = 1.606	12,809,668
LINGO	30	4 sec	x = 344.24 y = 1.8819	12,819,650

16) **MINING PROBLEM** (Taylor 1986)

Minimize: $z = 70.0035HL + 2333.33L^{-1} + 3333.33H^{-1} + 8333.33H^{-1}L^{-1}$

Optimal solution: z = 3032
H = 4.899
L = 3.429

Initial values: H = 1
L = 1
γ = .0000000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	31	< 1 sec	H = 4.899 L = 3.430	3032.92
MULTICON	97	16 sec	H = 4.899 L = 3.430	3032.92
MINOS	8	9 sec	H = 4.899 L = 3.430	3032.92
LINGO	8	1 sec	H = 4.898 L = 3.432	3032.92

17) OPTIMUM BITCYCLE SELECTION PROBLEM (Woolsey 1975)

Minimize: $z = 5000T^{.5} + 25000T^{-.5}$

Optimal solution: z = 22,360.67
T = 5

Initial values: T = 1
γ = .000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	6	< 1 sec	T = 5.00	22,360.68
MULTICON	2	< 1 sec	T = 5.00	22,360.68
MINOS	1	8 sec	T = 5.00	22,360.68
LINGO	2	< 1 sec	T = 5.00	22,360.68

18) STEAMPIPE INSULATION PROBLEM (Schweyer 1955)

Minimize: $z = 30s + 100s^{-1} + 40$

Optimal solution: z = 149.54
s = 1.826

Initial values: s = 1
γ = .000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	5	< 1 sec	s = 1.826	149.54
MULTICON	2	< 1 sec	s = 1.826	149.54
MINOS	4	9 sec	s = 1.826	149.54
LINGO	2	< 1 sec	s = 1.826	149.54

19) SPACE SHUTTLE DESIGN PROBLEM (Ratliff 1986)

Minimize:

$$z = 11.8609822x^{.470} + 441.1192843x^{-.146}$$
$$+3.218347592x^{.648} + 1467706.463x^{.568}$$
$$+1040x + 0.077708883x^{.736} + 23.68803092x^{-.229}$$

Optimal solution: z = 4,319.55
x = .00000237

Initial values: x = 1
γ = .00000000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	5	< 1 sec	.00000237	4,319.55
MULTICON	7	1 sec	.00000237	4,319.55
MINOS	1	10 sec	.00000237	4,319.55
LINGO	7	1 sec	.00000237	4,319.55

20) GEAR TRAIN INERTIA PROBLEM (Ravindran *et al* 1983)

Minimize: $z = .1\left[12 + x^2 + \frac{1+y^2}{x^2} + \frac{x^2y^2+100}{(xy)^4}\right]$

This can be rewritten as:

Minimize: $z = 1.2 + .1x^2 + .1x^{-2} + .1x^{-2}y^2 + .1x^{-2}y^{-2} + 10x^{-4}y^{-4}$

Optimal solution: z = 1.74415
x = 1.74345
y = 2.02969

Initial values: x = 1
y = 1
γ = .000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	9	< 1 sec	x = 1.7435 y = 2.0297	1.7442
MULTICON	47	8 sec	x = 1.6808 y = 2.041	1.7461
MINOS	9	10 sec	x = 1.743 y = 2.030	1.7442
LINGO	8	1 sec	x = 1.7435 y = 2.0297	1.7442

21) WESSELS PROBLEM 1 (Wessels 1989)

Minimize: $z = 5xy + 7x + 8y + 4x^{-2} + 8y^{-2}$

Optimal solution: z = 31.5686
x = 0.862787
y = 1.09121

Initial values: x = 1
y = 1
γ = .000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	5	< 1 sec	x = 0.8628 y = 1.0912	31.5686
MULTICON	15	3 sec	x = 0.8628 y = 1.0912	31.5686
MINOS	5	9 sec	x = 0.863 y = 1.091	31.5686
LINGO	2	< 1 sec	x = 0.8646 y = 1.0926	31.5687

22) REKLAITIS *et al* PROBLEM pg. 499 (1983)

Minimize: $z = 60x^{-3}y^{-2} + 50x^{3}y + 20x^{-3}y^{3}$

Optimal solution: z = 126.049
x = 1.10114
y = 0.944088

Initial values: x = 1
y = 1
γ = .000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	10	< 1 sec	x = 1.1011 y = 0.9441	126.049
MULTICON	72	11 sec	x = 1.1011 y = 0.9441	126.049
MINOS	5	9 sec	x = 1.101 y = 0.944	126.049
LINGO	8	1 sec	x = 1.1012 y = 0.9441	126.049

23) REKLAITIS *et al* PROBLEM pg. 531 (1983)

Minimize: $z = (xy)^{-1} + x^{.5} + y^{.75}$

Optimal solution: z = 2.88033
x = 1.76726
y = 0.851293

Initial values: x = 1
y = 1
γ = .000001

METHOD	# ITERATIONS	TIME	SOLUTION OBTAINED	Z VALUE
HARMONIC PROGRAMMING	5	< 1 sec	x = 1.7673 y = 0.8513	2.8803
MULTICON	28	4 sec	x = 1.7667 y = 0.8515	2.8803
MINOS	7	9 sec	x = 1.767 y = 0.851	2.8803
LINGO	5	1 sec	x = 1.7571 y = 0.8701	2.8805

Appendix B

The Program Listing

```
*******************************************************************
*
*   PROGRAM: HARMONIC PROGRAMMING
*
*   PURPOSE: This program solves unconstrained, multivariable, posynomial
*              problems by using the harmonic mean approximation and
*              condensation.
*
*   AUTHOR: Mark B. Pomeroy
*          CPT U.S. Army
*          Department of Mathematics and Computer Science
*          Colorado School of Mines
*
*   PROGRAMMED BY: Jason Kierstein
*          Department of Mathematics and Computer Science
*          Colorado School of Mines
*
*          Mark B. Pomeroy
*          CPT U.S. Army
*          Department of Mathematics and Computer Science
*          Colorado School of Mines
*
*   WRITTEN: June 1995
*
*   INPUTS: NVBLS: number of variables in the objective function
*          VNAME(K): name of the Kth variable
*          VARPWR(J,K): power in the Jth term of the Kth variable
*          COEF(J): coefficient of the Jth term
*          EPS: convergence tolerance
*          XBAR(I): starting value for the Ith variable
*          TERMS: number of terms in the objective function
*
*   OUTPUTS: DELTA(I): the optimal delta of the Ith term
*          ITER: the # of outerloop iterations to reach optimality
```

```
*          OBJ:  the optimal objective function value
*          XBAR(I):  the optimal value of the Ith variable
*
*    VARIABLES:  A:  upper triangular matrix
*            AA:  exponent matrix
*            COEF(J):  coefficient of the Jth term
*            COND:  an estimate of the condition number of A
*            CONDAA:  exponent matrix for the condensed obj. function
*            CONDCOEF(J):  the Jth term coefficient in the condensed
*                    obj. function
*            CONDELTA(J):  the delta for the Jth term in the
*                    condensed objective function
*            CONDPWR(J):  the Jth term power in the condensed
*                   objective function
*            CONVERGE:  reports whether the value has converged
*            COUNT:  a counter used when calculating XNEW
*            DD:  degree of difficulty
*            DELTA(I):  the calculated delta of the Ith term
*            EPS:  convergence tolerence
*            FLAG:  reports whether matrix A has a zero pivot
*            ITER:  the # of outerloop iterations to reach optimality
*            OBJ:  the harmonic mean objective function value
*            OBJCOND:  the H.M. value for the condensed obj. funct.
*            OBJ1ST:  the objective function value using XBAR(I)
*            MARK:  reports whether HP2 or HP3 is being used
*            NEWCOEF:  intermediate value when calculating OBJ1ST
*            NVBLS:  number of variables in the objective function
*            PVTIDX:  pivot vector keeping track of row interchanges
*            TEMP:  a vector used in calculating deltas
*            TEMPV:  intermediate value when calculating XNEW
*            TERMS:  number of terms in the objective function
*            VARPWR(J,K):  power in the Jth term of the Kth variable
*            VNAME(K):  name of the Kth variable
*            XBAR(I):  last value for the Ith variable
*            XNEW(I):  current value of the Ith variable
*            XVAL(I): an intermediate value for X(I) when calculating
*                 XNEW
*            VARINT:  the variable of interest
*******************************************************************
```

```
      program hp1

      implicit none

      real*8 obj,obj1st,eps,newcoef(50),condpwr(2),objcond,newterm(50)
      real*8 coef(50),varpwr(50,20),xval(20),xbar(20),xnew(20)
      real*8 delta(50),tempv,condelta(2),condaa(2,2),condcoef(2)
      double precision aa(20,20),temp(20),cond
      integer i,j,k,nvbls,terms,varint,term,count,pvtidx(20),flag
      integer converge, dd, mark,iter
      character*10 vname(20)
      character*1 rerun

      common iter

      print*, 'This program optimizes multivariable, unconstrained'
      print*, '0-dd nonlinear programming problems using the.'
      print*, 'harmonic mean approximation.'
      print*, ' '
      print*, 'The program is capable of handling functions with 20'
      print*, 'variables and 50 terms.'
      print*, ' '
      print*, 'Variable names must be no greater the 10 characters.'
      print*, ' '
      print*, 'Enter the number of variables in the problem: '
      read*, nvbls
**************************************************************

** INPUT THE VARIABLE NAMES

 4  format (1x,'Enter variable name ',i2,' ')
      do 6 i = 1,nvbls
       print 4,i
       read*, vname(i)
 6  continue

      print*, ' '
      print*, 'Enter the number of terms in the problem: '
      read*, terms
```

```
***********************************************************************

** INPUT THE COEFFICIENTS AND VARIABLE POWERS

      print*, ' '
 11  format (1x,'For term ',i2,' enter')
 12  format (1x,'The power on ',a10,' ')
      do 13 j = 1,terms
       print 11, j
       print*, 'The coefficient: '
       read*, coef(j)
       do 14 k = 1,nvbls
        print 12, vname(k)
        read*, varpwr(j,k)
 14   continue
 13  continue

 1  print*, ' '
      print*, 'Enter a convergence tolerance .xxxxx '
      read*, eps

    print*, ' '
    print*, 'We now need to enter a starting value for each of the'
    print*, 'variables.'
    print*, ' '
    do 18 i = 1, nvbls
       print 20, 'Enter the starting value for variable ',i,' '
     read *, xbar(i)
 18  continue
 20  format(1x,a,i2,a)

      dd = terms - nvbls - 1
      iter = 0

      if((dd.gt.0).and.(nvbls.eq.1)) then
        mark = 0
        call hp2(coef,varpwr,xbar,terms,eps,obj,mark,1)
        goto 157
      endif
      if((dd.gt.0).and.(nvbls.gt.1)) then
```

```
          call hp3(coef,varpwr,terms,nvbls,xbar,eps,obj)
          goto 157
        endif
*******************************************************************

** BUILD THE RULE 2 MATRIX

    do 25 i = 1,terms
      aa(1,i) = 1
 25 continue

    do 30 i = 2,nvbls+1
      do 35 j = 1,terms

              aa(i,j) = varpwr(j,i-1)

 35     continue
 30 continue

        delta(1) = 1

    do 40 i = 2,nvbls+1
          delta(i) = 0.0
 40 continue
*******************************************************************

** CALCULATE THE DELTAS

        call factor(aa,20,terms,cond,pvtidx,flag,temp)

** CHECK FOR VALID OUTPUT FROM FACTOR SUBROUTINE

        if(flag.gt.0) then
          print*,'Zero pivot in delta calculation'
        endif
        if(flag.lt.0) then
          print*,'Input error...check problem size'
        endif

        call solve(aa,20,terms,pvtidx,delta)
```

```
************************************************************************

** CALCULATE THE 1ST OBJECTIVE FUNCTION VALUE

 42  obj1st = 0.0
       do 45 i = 1,terms
          newcoef(i) = 1.0
          do 47 j = 1,nvbls
                newcoef(i) = newcoef(i) * xbar(j)**varpwr(i,j)
 47  continue
          obj1st = obj1st + newcoef(i)*coef(i)
 45  continue

************************************************************************

       call harmonic_mean(delta,coef,xbar,varpwr,obj,terms,nvbls)

************************************************************************

** CALCULATE NEW X's

       varint = 1
       term = 1

       do 141 i = 1,nvbls
          xnew(i) = -1
141  continue

145  if (varpwr(term,varint).ne.0) then
        count = 0
        do 153 j = 1,nvbls
          if((varpwr(term,j).eq.0).and.(j.ne.varint)) then
                count = count + 1
          endif
153  continue
          if (count .eq. nvbls-1) then
               xnew(varint)=(obj*delta(term)/coef(term))**(1/varpwr(term,va
   +rint))
               if(varint.lt.nvbls) then
```

```
            varint = varint + 1
            term = 1
            goto 145
           elseif(varint.eq.nvbls) then
            goto 190
           endif
      else
           if(term.lt.terms) then
            term = term+1
            goto 145
           endif
      endif

    else
      if(term.lt.terms) then
           term = term + 1
           goto 145
      endif
    endif

    do 170 i = 1,nvbls
     if(xnew(i) .eq. -1) then
      xval(i) = xbar(i)
     else
      xval(i) = xnew(i)
     endif
170 continue

    do 178 j = 1,terms
     tempv = 1.0
     do 179 i = 1,nvbls
      if(i.ne.varint) then
       tempv = tempv*xval(i)**varpwr(j,i)
      endif
      newcoef(j) = tempv*coef(j)
179   continue
178 continue

    call condense(newcoef,varpwr,xval,terms,varint,condcoef,condpwr)
```

```
      condaa(1,1) = 1.
      condaa(1,2) = 1.
      condaa(2,1) = condpwr(1)
      condaa(2,2) = condpwr(2)

      call factor(condaa,2,2,cond,pvtidx,flag,temp)

** CHECK FOR VALID OUTPUT FROM FACTOR SUBROUTINE

      if(flag.gt.0) then
        print*,'Zero pivot in delta calculation'
      endif
      if(flag.lt.0) then
        print*,'Input error...check problem size'
      endif

      condelta(1) = 1.
      condelta(2) = 0.
      call solve(condaa,2,2,pvtidx,condelta)

      call harmonic_mean(condelta,condcoef,xval(varint),condpwr,objcond,
     +2,1)

      xnew(varint)=(objcond*condelta(1)/condcoef(1))**(1/condpwr(1))
      if(varint.lt.nvbls) then
        varint = varint + 1
        goto 145
      elseif(varint.lt.nvbls) then
        goto 190
      endif

190   converge = 1
      i = 1
      do while ((converge.eq.1).and.(i.le.nvbls))
        call compare_values(xbar(i),xnew(i),eps,converge)
        i = i+1
      enddo

      if (converge.eq.1) then
        call compare_values(obj,obj1st,eps,converge)
```

```
      endif

      if(converge.eq.0) then
        do 195 i = 1,nvbls
              xbar(i) = xnew(i)
195     continue
      iter = iter + 1
      goto 42
      endif

      iter = iter + 1

157   print*,' '
      do 160 i = 1,nvbls
        print *, 'Optimal value of ',vname(i),' is ', xbar(i)
160   continue

      print*,
      print*, 'Optimal Objective Function Value is ',obj
      print*,
      print*,'Number of Iterations = ',iter

**********************************************************************

** PRINT DELTAS

      do 19 i = 1,terms
        newterm(i) = 1.0
19    continue

      do 22 i = 1,terms
       do 23 j = 1,nvbls
        newterm(i) = newterm(i)*xbar(j)**varpwr(i,j)
23    continue
      delta(i) = coef(i) * newterm(i) / obj
22    continue

      print*,
      do 50 i = 1, terms
        print 48,i,delta(i)
```

```
 50 continue
 48 format(1x,'% contribution at optimality for term ',i2,' = ',f6.4)

      print*,
      print*,'Would you like to rerun the problem with different'
      print*,'starting values and/or epsilon? (y or n) '
      read*, rerun

      if(rerun.eq.'y') then
        goto 1
      endif


      stop
      end

*********************************************************************

** SUBROUTINE TO COMPARE XOLD, XNEW, AND OBJ FCN VALUES

      subroutine compare_values(first,second,eps,converge)

      real*8 first,second,eps,diff
      integer converge

      diff = abs(first-second)
      if(diff.lt.eps) then
        converge = 1
      else
        converge = 0
      endif
      return
      end

*********************************************************************

** SUBROUTINE TO CONDENSE PROBLEM TO 0 DD

      subroutine condense(newcoef,varpwr,xval,terms,varint,condcoef,
     +condpwr)
```

```
      real*8 newcoef(50),varpwr(50,20),xval(20),condcoef(2)
      real*8 condpwr(2),wp(50),wn(50),sdpos,sdneg
      integer varint,terms,i

      sdpos = 0.
      sdneg = 0.
      condcoef(1) = 1.
      condcoef(2) = 1.
      condpwr(1) = 0.
      condpwr(2) = 0.

      do 300 i = 1,terms

       if(varpwr(i,varint).gt.0) then
        sdpos = sdpos+newcoef(i)*xval(varint)**varpwr(i,varint)
       elseif (varpwr(i,varint).lt.0) then
        sdneg = sdneg+newcoef(i)*xval(varint)**varpwr(i,varint)
       else
       endif

300   continue

      do 310 i = 1,terms
       if(varpwr(i,varint).gt.0) then
        wp(i) = (newcoef(i)*xval(varint)**varpwr(i,varint))/sdpos
       elseif(varpwr(i,varint).lt.0) then
        wn(i) = (newcoef(i)*xval(varint)**varpwr(i,varint))/sdneg
       else
       endif
310   continue

      do 320 i = 1,terms
       if(varpwr(i,varint).gt.0) then
        condcoef(1) = condcoef(1)*(newcoef(i)/wp(i))**wp(i)
        condpwr(1) = condpwr(1)+varpwr(i,varint)*wp(i)
       elseif(varpwr(i,varint).lt.0) then
        condcoef(2) = condcoef(2)*(newcoef(i)/wn(i))**wn(i)
        condpwr(2) = condpwr(2)+varpwr(i,varint)*wn(i)
       else
```

```
        endif
 320  continue
        return
        end

**********************************************************************

** SUBROUTINE TO CALCULATE THE HARMONIC MEAN

        subroutine harmonic_mean(delta,coef,xbar,varpwr,zh,terms,nvbls)

        implicit none
        real*8 delta(50),coef(50),xbar(20),varpwr(50,20),zh
        real*8 term(50),newcoef(50)
        integer i,j,terms,nvbls

        do 45 i = 1,terms
           newcoef(i) = 1.0
           do 47 j = 1,nvbls
                 newcoef(i) = newcoef(i) * xbar(j)**varpwr(i,j)
 47        continue
           term(i) = newcoef(i)*coef(i)
 45  continue

        zh = 0.0

        do 80 i = 1,terms
           zh = zh + (delta(i)**2)/term(i)
 80  continue
        zh = zh**(-1)
        end

**********************************************************************

** SUBROUTINE TO FACTOR THE COEFFICIENT MATRIX

     SUBROUTINE FACTOR(A,MAXROW,NEQ,COND,PVTIDX,FLAG,TEMP)
*
     INTEGER MAXROW,NEQ,PVTIDX(*),FLAG
     DOUBLE PRECISION A(MAXROW,*),COND,TEMP(*)
```

```
*
* FACTOR decomposes the matrix A using Gaussian elimination
* and estimates its condition number. FACTOR may be used in
* conjunction with SOLVE to solve A*x=b.
*
* Input variables:
*   A      = matrix to be triangularized.
*   MAXROW = maximum number of equations allowed; the declared row
*            dimension of A.
*   NEQ    = actual number of equations to be solved; NEQ cannot
*            exceed MAXROW.
* Output variables:
*   A      = the upper triangular matrix U in its upper portion
*            and a permuted version of a lower triangular matrix
*            I-L such that (permutation matrix)*A = L*U; a
*            record of interchanges is kept in PVTIDX.
*   FLAG   = an integer variable that reports whether or not the
*            matrix A has a zero pivot. A value of FLAG = 0
*            means all pivots were nonzero; if positive, the
*            first zero pivot occurred at equation FLAG and the
*            decomposition could not be completed. If FLAG = -1
*            then there is an input error (NEQ or MAXROW not positive
*            or NEQ > MAXROW).
*   COND   = an estimate of the condition number of A (unless
*            FLAG is nonzero).
*   PVTIDX = the pivot vector which keeps track of row inter-
*            changes; also,
*                PVTIDX(NEQ) = (-1)**(number of interchanges).
*   TEMP   = a vector of dimension NEQ used for a work area.
*
* The determinant of A can be obtained on output from
*   DET(A) = PVTIDX(NEQ) * A(1,1) * A(2,2) * ... * A(NEQ,NEQ).
*
* Declare local variables and initialize:
      DOUBLE PRECISION ANORM,DNORM,T,YNORM
      INTEGER I,J,K,M
      DOUBLE PRECISION ZERO,ONE
      DATA ZERO/0.D0/,ONE/1.D0/
*
      IF ((NEQ .LE. 0) .OR. (MAXROW .LE. 0) .OR. (NEQ .GT. MAXROW)) THEN
```

```
          FLAG = -1
          RETURN
        ENDIF
        FLAG = 0
        COND = ZERO
        PVTIDX(NEQ) = 1
        IF (NEQ .EQ. 1) THEN
*
*         NEQ = 1 is a special case.
*
          IF (A(1,1) .EQ. ZERO) THEN
            FLAG = 1
          ELSE
            COND = ONE
          ENDIF
          RETURN
        ENDIF
*
*     Compute 1-norm of A for later condition number estimation.
*
        ANORM = ZERO
        DO 15 J = 1,NEQ
          T = ZERO
          DO 10 I = 1,NEQ
            T = T+ABS(A(I,J))
   10     CONTINUE
          ANORM = MAX(T,ANORM)
   15 CONTINUE
*
*     Gaussian elimination with partial pivoting.
*
        DO 40 K = 1,NEQ-1
*
*         Determine the row M containing the largest element in
*         magnitude to be used as a pivot.
*
          M = K
          DO 20 I = K+1,NEQ
            IF (ABS(A(I,K)) .GT. ABS(A(M,K))) M = I
   20     CONTINUE
```

```
*
*      Check for a nonzero pivot; if all possible pivots are zero,
*      matrix is numerically singular.
*
       IF (A(M,K) .EQ. ZERO) THEN
         FLAG = K
         RETURN
       ENDIF
       PVTIDX(K) = M
       IF (M .NE. K) THEN
*
*        Interchange the current row K with the pivot row M.
*
         PVTIDX(NEQ) = -PVTIDX(NEQ)
         DO 25 J = K,NEQ
           T = A(M,J)
           A(M,J) = A(K,J)
           A(K,J) = T
  25     CONTINUE
       ENDIF
*
*      Eliminate subdiagonal entries of column K.
*
       DO 35 I = K+1,NEQ
         T = A(I,K)/A(K,K)
         A(I,K) = -T
         IF (T .NE. ZERO) THEN
           DO 30 J = K+1,NEQ
             A(I,J) = A(I,J)-T*A(K,J)
  30       CONTINUE
         ENDIF
  35   CONTINUE
  40 CONTINUE
*
     IF (A(NEQ,NEQ) .EQ. ZERO) THEN
       FLAG = NEQ
       RETURN
     ENDIF
*
*    Estimate the condition number of A.
```

```
*
   DO 50 K = 1,NEQ
     T = ZERO
     DO 45 I = 1,K-1
       T = T+A(I,K)*TEMP(I)
 45   CONTINUE
     TEMP(K) = -(SIGN(ONE,T)+T)/A(K,K)
 50 CONTINUE
   DO 60 K = NEQ-1,1,-1
     T = ZERO
     DO 55 I = K+1,NEQ
       T = T+A(I,K)*TEMP(K)
 55   CONTINUE
     TEMP(K) = T
     M = PVTIDX(K)
     IF (M .NE. K) THEN
       T = TEMP(M)
       TEMP(M) = TEMP(K)
       TEMP(K) = T
     ENDIF
 60 CONTINUE
   DNORM = ZERO
   DO 65 I = 1,NEQ
     DNORM = DNORM+ABS(TEMP(I))
 65 CONTINUE
   CALL SOLVE(A,MAXROW,NEQ,PVTIDX,TEMP)
   YNORM = ZERO
   DO 70 I = 1,NEQ
     YNORM = YNORM+ABS(TEMP(I))
 70 CONTINUE
   COND = ANORM*YNORM/DNORM
   RETURN
     END
*********************************************************************

** SUBROUTINE TO SOLVE THE LINEAR SYSTEM

   SUBROUTINE SOLVE(A,MAXROW,NEQ,PVTIDX,B)
*
   INTEGER MAXROW,NEQ,PVTIDX(*)
```

```
      DOUBLE PRECISION A(MAXROW,*),B(*)
*
* SOLVE solves the linear system A*x=b using the factorization
* obtained from FACTOR. Do not use SOLVE if a zero pivot has
* been detected in FACTOR.
*
* Input variables:
*   A      = an array returned from FACTOR containing the
*            triangular decomposition of the coefficient matrix.
*   MAXROW = as in FACTOR.
*   NEQ    = number of equations to be solved.
*   PVTIDX = vector of information about row interchanges obtained
*             from FACTOR.
*   B      = right hand side vector b.
* Output variables:
*   B      = solution vector x.
*
* Local variables:
      INTEGER I,J,K,M
      DOUBLE PRECISION T
*
*   Forward elimination.
*
      IF (NEQ .GT. 1) THEN
        DO 20 K = 1,NEQ-1
          M = PVTIDX(K)
          T = B(M)
          B(M) = B(K)
                  B(K) = T
          DO 10 I = K+1,NEQ
            B(I) = B(I)+A(I,K)*T
   10     CONTINUE
   20   CONTINUE
*
*     Back substitution.
*
        DO 40 I = NEQ,1,-1
          DO 30 J = I+1,NEQ
            B(I) = B(I)-A(I,J)*B(J)
   30     CONTINUE
```

```
      B(I) = B(I)/A(I,I)
 40   CONTINUE
      ELSE
     B(1) = B(1)/A(1,1)
    ENDIF
    RETURN
    END
***********************************************************************

** SUBROUTINE TO SOLVE MULTIPLE DD SINGLE VARIABLE PROBLEMS

      subroutine hp2(coef,varpwr,xbar,terms,eps,objcond,mark,varint)

      real*8 eps,condpwr(2),objcond
      real*8 coef(50),varpwr(50,20),xbar(20),xnew(20)
      real*8 condelta(2),condaa(2,2),condcoef(2)
      double precision temp(20),cond
      integer terms,pvtidx(20),flag,varint
      integer converge, mark,iter

      common iter
      iter = 0

300   call condense(coef,varpwr,xbar,terms,1,condcoef,condpwr)

      condaa(1,1) = 1.
      condaa(1,2) = 1.
      condaa(2,1) = condpwr(1)
      condaa(2,2) = condpwr(2)

      call factor(condaa,2,2,cond,pvtidx,flag,temp)

      condelta(1) = 1.
      condelta(2) = 0.
      call solve(condaa,2,2,pvtidx,condelta)

      call harmonic_mean(condelta,condcoef,xbar(varint),condpwr,objcond,
     +2,1)

      xnew(varint)=(objcond*condelta(1)/condcoef(1))**(1/condpwr(1))
```

```
      if(mark.eq.1) then
         xbar(varint) = xnew(varint)
      endif

      if(mark.eq.0) then
        iter = iter + 1
        call compare_values(xbar(varint),xnew(varint),eps,converge)

        if (converge.eq.0) then
          xbar(varint) = xnew(varint)
          goto 300
        endif
      endif

      return
      end

*******************************************************************

** SUBROUTINE TO SOLVE MULTIPLE VARIABLE, MULTIPLE DD PROBLEMS

      subroutine hp3(coef,varpwr,terms,nvbls,xbar,eps,obj)

      real*8 obj,eps,newcoef(50),xval(20),condaa(2,2),objcond
      real*8 coef(50),varpwr(50,20),xbar(20),xnew(20),temp(20)
      real*8 tempv,condpwr(2),condcoef(2),cond,condelta(2)
      integer i,j,nvbls,terms,varint,pvtidx(20)
      integer converge,flag,iter

      common iter

404   varint = 1
      do 401 i = 1,nvbls
         xnew(i) = -1
401   continue

405    do 402 i = 1,nvbls
        if(xnew(i).eq.-1) then
          xval(i) = xbar(i)
```

```
        else
          xval(i) = xnew(i)
        endif
 402  continue

      do 478 j = 1,terms
        tempv = 1.0
        do 479 i = 1,nvbls
          if(i.ne.varint) then
            tempv = tempv*xval(i)**varpwr(j,i)
          endif
          newcoef(j) = tempv*coef(j)
 479    continue
 478  continue

      call condense(newcoef,varpwr,xval,terms,varint,condcoef,condpwr)

      condaa(1,1) = 1.
      condaa(1,2) = 1.
      condaa(2,1) = condpwr(1)
      condaa(2,2) = condpwr(2)

      call factor(condaa,2,2,cond,pvtidx,flag,temp)

      condelta(1) = 1.
      condelta(2) = 0.
      call solve(condaa,2,2,pvtidx,condelta)

      call harmonic_mean(condelta,condcoef,xval(varint),condpwr,objcond,
     +2,1)

      xval(varint)=(objcond*condelta(1)/condcoef(1))**(1/condpwr(1))

      if(varint.eq.nvbls) then
         converge = 1
         iter = iter + 1
         do 400 i = 1,nvbls
               if(converge.eq.1) then
                 xnew(varint) = xval(varint)
                 call compare_values(xbar(i),xnew(i),eps,converge)
```

```
            endif
400      continue
           if(converge.eq.0) then
                  do 410 i = 1,nvbls
                        xbar(i) = xnew(i)
410         continue
           endif
        else
           xnew(varint) = xval(varint)
           varint = varint + 1
           goto 405
        endif

        if(converge.eq.0) then
           goto 404
        endif

        obj = 0.0
        do 445 i = 1,terms
           newcoef(i) = 1.0
           do 447 j = 1,nvbls
                 newcoef(i) = newcoef(i) * xbar(j)**varpwr(i,j)
447  continue
           obj = obj + newcoef(i)*coef(i)
445  continue

        return
        end
```

Appendix C

Sample Computer Run for Harmonic Programming

C:\LP77>HP1

This program optimizes multivariable, unconstrained
0-dd, nonlinear programming problems using the
harmonic mean approximation.

The program is capable of handling functions with 20
variables and 50 terms.

Variable names must be no greater than 10 characters.

Enter the number of variables in the problem: 2

Enter variable 1's name: x1

Enter variable 2's name: x2

Enter the number of terms in the problem: 3

For term 1 enter
the coefficient: 78

The power on x1: 1

The power on x2: 0

For term 2 enter
the coefficient: 27

The power on x1: -1

The power on x2: -1

For term 3 enter
the coefficient: 58

The power on x1: 0

The power on x2: 1

Enter a convergence tolerence .xxxxx .000001

We now need to enter a starting value for each of the variables.

Enter the starting value for variable 1: 1

Enter the starting value for variable 2: 1

Optimal value of x1 is 0.636112864970562
Optimal value of x2 is 0.855462128753514

Optimal Objective Function Value is 148.850412159240

Number of Iterations = 5

% contribution at optimality for term 1 = 0.3333
% contribution at optimality for term 2 = 0.3333
% contribution at optimality for term 3 = 0.3333

Would you like to rerun the problem with different starting values and/or epsilon? (y or n)