United States
Department of
Agriculture

Statistical
Reporting
Service

Statistical
Research
Division

SRS Staff Report
Number SF&SRB-89

January 1986

An Optimum Allocation Algorithm for Multivariate Surveys

James Bethel

AN OPTIMUM ALLOCATION ALGORITHM FOR MULTIVARIATE SURVEYS. By James Bethel, Statistical Research Division, Statistical Reporting Service, U.S. Department of Agriculture, Washington, D.C. 20250. Staff Report No. SRS-SRB-89.

ABSTRACT

Optimal sample allocation, when multivariate data are collected, is a difficult problem and can be time-consuming even on large computers. This paper developes and presents a relatively simple algorithm which is readily programmable and which appears to work very efficiently, converging quickly even on small computers. A complete description of the algorithm is given, along with a proof of convergence, and a discussion of methods used by the author in implementing the algorithm. PASCAL source code for a program written by the author is provided.

Keywords: Convex programming, sample design, stratified sampling.

CONTENTS

SUMMARY

Optimum allocation is a useful, if not invaluable, tool in sample design. If there is only one variable of interest, the problem is easily solved. When more than one variable is being observed, there are no closed-form optimum solutions and some form of iterative computer algorithm must be used. Such programs are available but generally only on large computer systems (such as Martin Marietta) which are somewhat expensive to use, especially when compared with programs that run on small in-house computing equipment. Specifically, this algorithm is recommended as an efficient program for the Statistical Reporting Service to use in multivariate optimum sample allocation problems using an in-house computer.

This paper documents a program developed by the author to solve such multivariate problems. The paper is written at a technical level and is not intended for general dissemination. The algorithm on which it is based is simpler and seems to be more efficient than others designed for this purpose. In particular, the program runs well on a Zilog System 8000, a microcomputer which is available to many statisticians in the Statistical Reporting Service. The algorithm is simple enough that the program can be easily adapted to solving many different types of allocation problems. The author used the program extensively in evaluating sample designs for the Integrated Survey Program (ISP), the work which originally motivated the developement of the algorithm (Bethel, 1985). The program has also been used for studying multiple frame estimators and for allocations involving double-sampling regression estimators (Williams, 1986).

In this report, the algorithm is described and shown to converge to the optimum solution. The convergence is automatic, in the sense that the starting values are picked by the algorithm and the convergence does not depend on the values chosen. There are, however, certain parameters in the program which are discussed with some informal motivations given for the values used by the author. Finally, PASCAL source code is given for a program written by the author to implement the algorithm.

AN OPTIMUM ALLOCATION ALGORITHM FOR MULTIVARIATE SURVEYS

James Bethel

INTRODUCTION

The problem of optimal sample allocation for multipurpose surveys can be viewed more generally as a problem in convex programming and, as such, there are many ways to obtain a numerical solution. Huddleston, Claypool and Hocking (1970) have applied a nonlinear programming method devised by Hartley and Hocking (1963) to this problem, and Kokan (1963) has discussed some standard nonlinear programming techniques with respect to optimal allocation. While these and other, more general methods are available, most of them are difficult to program and computationally burdensome, and not all are guaranteed to converge. In this paper, an algorithm is developed which is relatively simple to program and which converges quickly, even on small computers.

Consider the case of stratified random sampling. Suppose it is required that the j-th variable, $1 \leq j \leq p$, satisfy

$$\operatorname{var}(\bar{y}_j) = \sum_{i=1}^{L} w_i^2 S_{ij}^2 \left(\frac{1}{n_i} - \frac{1}{N_i}\right) \leq v_j.$$

Let

$$x_i = \begin{cases} \dfrac{1}{n_i} & \text{if } n_i \geq 1 \\ \infty & \text{otherwise.} \end{cases}$$

Assume the cost function

$$C = \sum_{i=1}^{L} c_h n_h = \sum_{i=1}^{L} \frac{c_i}{x_i}.$$

(Fixed costs are irrelevant since these do not affect the minimization problem.) Now the problem reduces to minimizing

$$C = C(x) \tag{1}$$

subject to the constraints*

$$\sum_{i=1}^{L} a_{ji} x_i \leq 1 , \qquad 1 \leq j \leq p \tag{2}$$

where

$$a_{ji} = \left[w_i^2 S_{ij}^2\right] / \left[v_j + \sum_{i=1}^{L} \frac{w_i^2 S_{ij}^2}{N_i}\right].$$

The discussion will be limited to this allocation model, since Kokan (1963) shows how it can be adapted to cover virtually any sampling situation. The algorithm is described in the next section. The proof of convergence, which is presented in the third section, is the main focus of this paper, but it is not essential for the discussion of applications. PASCAL code for implementing the algorithm is given in the Appendix.

THE ALGORITHM

Consider this informal argument: For fixed values of k, the set

$$S_k = \{x: \sum \frac{c_i}{x_i} = k\} \tag{3}$$

forms a convex hyperboloid, while the set

---

*Ideally, one would also want

$$N_i^{-1} \leq x_i \leq 1 , \qquad 1 \leq i \leq L.$$

An automatic method for incorporating this constraint into the problem has not been developed, but the method of Cochran (1977), p. 104, can easily be used.

$$F = \{x: a_j'x \leq 1,\ 1 \leq j \leq p\} \tag{4}$$

forms a convex polygon below S_k. As k increases, S_k moves downward toward the upper boundary of the feasible region F and the point where these sets meet is the optimal solution to (1) and (2). This is depicted in Figure 1.

For any hyperplane

$$H = \{x: a'x = 1\}. \tag{5}$$

Kokan and Khan (1967) show that H and S_k are tangent (for some suitable k) at the point t, where

$$t_i = \begin{cases} (c_i\ a_i)^{1/2} \ /\ (a_i \sum_{i=1}^{L} (c_i\ a_i)^{1/2}) & \text{if } a_i \neq 0 \\ \infty & \text{otherwise.} \end{cases} \tag{6}$$

Consider the $a_j = (a_{j1}, a_{j2}, \ldots, a_{jL})'$, as defined by (2). Let $H_j = \{x: a_j'x = 1\}$ and suppose that $t_j = (t_{j1}, t_{j2}, \ldots, t_{jL})'$ is the point where H_j and S_k are tangent. If $t_j \in F$, then, as Kokan and Khan (1967) show, t_j is the optimal solution to (1). Unfortunately, this is rarely the case.

Suppose that $H = \{x: a'x = 1\}$ and $t = t(H)$ is the point where, for some suitable k, H is tangent to S_k. The cost C(t) can be written as a function of the coefficients a_i:

$$C(t) = \sum_{i=1}^{L} \frac{c_i}{t_i} \tag{7}$$

$$= \sum_{i=1}^{L} c_i \ /\ \left[(c_i\ a_i)^{1/2} \ /\ (a_i \sum_{i=1}^{L} (c_i\ a_i)^{1/2})\right]$$

$$= \sum_{i=1}^{L} (c_i\ a_i)^{1/2} \sum_{i=1}^{L} (c_i\ a_i)^{1/2}$$

$$= \left[\sum_{i=1}^{L} (c_i\ a_i)^{1/2}\right]^2.$$

For convenience write

$$G(H) = C(t(H)) = C(t). \tag{8}$$

The algorithm begins by selecting one of the H_j as an initial value $H^{(1)} = \{x: a^{(1)\prime}x = 1\}$. For example take

$$H^{(1)} = H_1.$$

Now choose $H^{(2)}$ to satisfy

$$G(H^{(2)}) = \max_{0\leq\beta\leq 1} \left[\sum_i (c_i(\beta a_i^{(1)} + (1-\beta)a_{2i})^{1/2}\right]. \tag{9}$$

In a sense, $H^{(2)}$ is the convex combination of $H^{(1)}$ and H_2 which maximizes G. To find $H^{(3)}$, repeat this process with H_3, replacing $a_i^{(1)}$ with $a_i^{(2)}$ and a_{2i} with a_{3i} in formula (9).

In general choose $H^{(n+1)}$ to satisfy

$$G(H^{(n+1)}) = \max_{0\leq\beta\leq 1} \left[\sum (c_i(\beta a_i^{(n)} + (1-\beta)a_{j_n i})^{1/2}\right] \tag{10}$$

where $j_n = n+1 \pmod p$. For the purpose of this discussion, one iteration will consist of calculating $H^{(1)}, H^{(2)}, \ldots, H^{(p)}$.

CONVERGENCE OF THE ALGORITHM

The algorithm defined by (10) has the property that it is guaranteed to converge, regardless of the initial starting value. (Notice that the ordering of the H_j has not been specified in any way, so that the choice of H_1 for the starting value was arbitrary.)

Suppose that x^* satisfies

$$C(x^*) \leq C(x) \text{ for all } x \in F \tag{11}$$

and let H^* be the hyperplane that is tangent to $S_{C(x^*)}$ at x^*. Kokan and Khan (1967) establish the existence and uniqueness of x^*. Let $x^{(n)}$ be the point where $H^{(n)}$ and $S_{x^{(n)}}$ intersect. It will be shown that

$$\lim_n x^{(n)} = x^*.$$

The proof is based on three results. these are stated now, deferring proof until after the presentation of the main result.

Theorem 1. Suppose that $p_1, p_2, \ldots, p_k \in R^n$, with $p_i \neq p_j$ whenever $i \neq j$, and that $f: D = \{x: x = \sum \gamma_j p_j,\ \gamma_j \geq 0,\ \sum \gamma_j = 1\} \to R^1$ is strictly concave. Then $f(z)$ is a maximum on D if

$$f(z) \geq f(z + \alpha(p_j - z))$$

for all $\alpha \in [0,1]$ and for each $j = 1,2,\ldots,k$.

Theorem 2. For any set of the form $H = \{x: \sum \alpha_j a_j' x = 1,\ \alpha_j \geq 0,\ \sum \alpha_j = 1\}$,

$$G(H) \leq G(H^*).$$

Theorem 3. There exist $\alpha_j \geq 0$, with $\sum_j \alpha_j = 1$, such that

$$H^* = \{x: \sum_j \alpha_j a_j' x = 1\}.$$

From (10) it is apparent that the algorithm searches among hyperplanes of the form $\{x: \sum \alpha_j a_j' x = 1\}$. Theorems 2 and 3 show that H^* has this form and, in fact, maximizes cost over all such hyperplanes. Theorem 1 shows that it is sufficient to search in pairwise fashion, maximizing the convex combination of the current hyperplane with each of the H_j in succession.

The main result is now proved:

Theorem 4. $\lim_n x^{(n)} = x^*$.

Proof: From (10) and Theorem 2

$$0 \leq G(H^{(n)}) \leq G(H^{(n+1)}) \leq G(H^*). \tag{12}$$

But note that

$$v = (N_1^{(-1)}, N_2^{(-1)}, \ldots, N_L^{(-1)})' \in F$$

(since then $\text{var}(\bar{y}_j) = 0$) thus, from (11),

$$G(H^*) \leq C(v) < \infty. \tag{13}$$

Therefore $G(H^{(n)})$ is a monotonic, bounded sequence and must have a finite limit.

Let H be any hyperplane of the form given in Theorem 2, and set $b_i = \sum \alpha_j a_j$. Notice that the function

$$G(H) = \sum_i (c_i b_i)^{1/2}$$

is strictly concave. This follows from

$$\frac{\partial^2 G(H)}{\partial b_r \partial b_s} = \begin{cases} \dfrac{-c_s^2}{4(c_s b_s)^{3/2}} & \text{if } r = s \\ 0 & \text{otherwise,} \end{cases}$$

which implies that the matrix of second partial derivatives is negative definite.

It now follows easily from Theorems 1 and 3 that

$$\lim_n G(H^{(n)}) = \max_\alpha \{G(H)\} = G(H^*). \tag{13}$$

Since a concave function is being maximized over a convex set there is a unique maximum (see Avriel, 1976, p. 94), thus

$$\lim_n H^{(n)} = H^*$$

and the result follows.

The rest of this section is given to the proofs of Theorems 1-3. It will be useful to establish some basic facts concerning convex functions. First of all, if f is a strictly convex function then, by Taylor's theorem

$$f(x) - f(y) = (x - y)'\nabla f(y) + \xi' Q \xi \tag{14}$$

(by strict convexity f has first and second partial derivatives). Furthermore the matrix Q is nonnegative definite so that

$$f(x) - f(y) \geq (x - y)'\nabla f(y). \tag{15}$$

It also follows from (14) that if $z = \alpha x + (1-\alpha)y$, where $0 \leq \alpha \leq 1$, then

$$f(z) - f(y) = \alpha(x-y)'\nabla f(y) + O(\alpha^2). \tag{16}$$

Finally, note that the hyperplane tangent to f(x) at a point c has the equation

$$y = f(c) + (x-c)'\nabla f(c).$$

If it is required that $y = k = f(c)$ then the plane tangent to the plane $S_k = S_{f(c)}$ has the equation

$$x'\nabla f(c) = c'\nabla f(c). \tag{17}$$

In particular,

$$x'\nabla C(x) = -\sum_i x_i \frac{c_i}{x_i^2} = -\sum_i \frac{c_i}{x_i} = -C(x). \tag{18}$$

Define h by

$$H^* = \{x: h'x = 1\}. \tag{19}$$

Then

$$h = -(C(x^*))^{-1}\left[\frac{c_1}{x_1^*}, \frac{c_2}{x_2^*}, \ldots, \frac{c_L}{x_L^*}\right]'. \tag{20}$$

Theorem 1. Suppose that $p_1, p_2, \ldots, p_k \in R^n$, with $p_i \neq p_j$ whenever $i \neq j$, and that $f: D = \{x: x = \sum \gamma_j p_j,\ \gamma_j \geq 0,\ \sum \gamma_j = 1\} \to R^1$ is strictly concave. Then f(z) is a maximum on D if

$$f(z) \geq f(z + \alpha(p_j - z)) \tag{21}$$

for any $\alpha \in [0,1]$ and for each $j = 1,2,\ldots,k$.

Proof: Using Taylor's theorem

$$f(z + \alpha(p_j - z)) = f(z) + \alpha(p_j - z)'\nabla f(z) + O(\alpha^2). \tag{22}$$

Combining (21) and (22), and letting α tend to 0,

$$(p_j - z)'\nabla f(z) \leq 0 \tag{23}$$

for each $j = 1,2,\ldots,k$. Suppose that $u \in D$. Since $-f$ is strictly convex, combining (15) and (23) yields

$$\begin{aligned} f(u) - f(z) &\leq (u - z)'\nabla f(z) \\ &\leq (\textstyle\sum \gamma_j p_j - z)'\nabla f(z) \\ &\leq (\textstyle\sum \gamma_j p_j - \sum \gamma_j z)'\nabla f(z) \\ &\leq \textstyle\sum \gamma_j (p_j - z)'\nabla f(z) \\ &\leq 0 \end{aligned}$$

and the result follows.

Theorem 2. For any set of the form $H = \{x: \sum \alpha_j a_j'x = 1,\ \alpha_j \geq 0,\ \sum \alpha_j = 1\}$,

$$G(H) \leq G(H^*).$$

Proof: Let t be the intersection of H and S_k. Note that C is strictly convex, so that, using (15),

$$G(H^*) - G(H) = C(x^*) - C(t) \geq (x^* - t)'\nabla C(t). \tag{24}$$

Notice that since H is tangent to C at t the equation for H is given by

$$x'\nabla C(t) = t'\nabla C(t)$$

so that

$$\sum \alpha_j a_j = (t'\nabla C(t))^{-1}\nabla C(t). \tag{25}$$

From (18)

$$t'\nabla C(t) = -C(t) < 0.$$

But notice that

$$1 \geq \sum \alpha_j a_j' x^*$$

which yields

$$t'\nabla C(t) \leq (t'\nabla C(t)) \sum \alpha_j a_j' x^*$$

$$= x^{*\prime}\nabla C(t).$$

It follows that

$$G(H^*) - G(H) \geq (x^* - t)'\nabla C(t) \geq 0.$$

Two preliminary results are necessary to prove Theorem 3. The first is proved in Kokan and Khan (1967). The second, due to Kuhn and Tucker (1951), is one of the fundamental results in nonlinear programming; this version is slightly restated from Theorem 4.39 of Avriel (1976).

Lemma 1. For some subset J of {1,2,...,p}

$$x^* \in I_J = \{x: a_j'x = 1,\ j \in J\}.$$

Lemma 2. Suppose that f and $g_1, g_2, \ldots, g_p$ are real-valued and differentiable functions on R^n, where f is convex and the g_j are concave. Let

$$F = \{x: g_j \geq 0,\ 1 \leq j \leq p\}.$$

Suppose that there is $v \in R^n$ such that

$$g_j(v) > 0$$

for $j = 1,2,\ldots,p$. If z satisfies

$$\min_F\{f(x)\} = f(z)$$

then there exist $\lambda_j \geq 0$ for which

$$\nabla f(z) - \sum_j \lambda_j \nabla g_j(z) = 0$$

and

$$\lambda_j g_j(z) = 0$$

for $j = 1,2,\ldots,p$.

Theorem 3. There exist $\alpha_j \geq 0$, with $\sum_j \alpha_j = 1$, such that

$$H^* = \{x: \sum_j \alpha_j a_j'x = 1\}.$$

Proof: Suppose that

$$H^* = \{x: h'x = 1\}.$$

In Lemma 2 take

$$f(x) = (-C(x^*))C(x)$$

$$v = (N_1^{(-1)}, N_2^{(-1)}, \ldots, N_L^{(-1)})',$$

$$g_j(x) = -(a_j'x - 1).$$

It follows that there exist $\lambda_j \geq 0$ such that

$$-C(x^*)\nabla C(x^*) + \sum_j \lambda_j a_j = 0$$

and

$$\lambda_j(a_j'x^* - 1) = 0, \qquad j = 1,2,\ldots p.$$

Suppose that $x^* \in J$ as in Lemma 1. If j is not in J then

$$a_j'x^* - 1 < 0$$

which implies that $\lambda_j = 0$. It follows from (20) that

$$h = \sum_{j \in J} \lambda_j a_j.$$

But if $j \in J$ then $a_j'x^* = 1$, thus

$$1 = h'x^* = \sum_{j \in J} \lambda_j a_j'x^*$$

$$= \sum_{j \in J} \lambda_j$$

The result follows by taking

$$\alpha_j = \begin{cases} \lambda_j & \text{if } j \in J \\ 0 & \text{otherwise.} \end{cases}$$

IMPLEMENTING THE ALGORITHM

As has been shown, the algorithm converges for any starting value $H^{(1)}$, but, intuitively, it makes sense to maximize $C(H^{(j)})$, since $x^{(1)}$ should be as close as possible to F. Thus take

$$H^{(1)} = H^{(j_0)}$$

where

$$C(H^{(j_0)}) \geq C(H^{(j)})$$

for all j. Since

$$g(\alpha) = \sum_i (c_i(\alpha a_i^{(n)} + (1-\alpha)a_{ji}))^{1/2}$$

is concave in α, a direct search for the maximum on [0,1] can be carried out. This was accomplished by using a small positive value β and finding

$$\max \{g(0), g(\beta), g(2\beta), \ldots, g(k\beta), \ldots, g(1)\}.$$

This was done by starting with g(0) and stopping when

$$g(k\beta) \geq g((k+1)\beta).$$

This is not as inefficient as it may appear. Even on problems with many constraints, there are usually only two or three which determine the solution; that is, most of the constraints will not be searched since any value $\beta > 0$ results in lowering the cost. Also, the value of β which maximizes g is usually quite small and almost always less than .5. Therefore this searching method wastes little time searching over unneccessary constraints which are and does not require many steps to find the optimal β, since this value is small.

In order for convergence to occur, β must be decremented. This was done at each iteration. That is, after finding

$$\max_k \left[\sum_i (c_i(k\beta a_i^{(n)} + (1-k\beta)a_{ji})^{1/2}\right] \quad (26)$$

where $k = 0,1,\ldots,[\beta^{-1}]$, for $j = 1,2,\ldots,p$, and β was decremented and the process was repeated until the convergence criterion was satisfied. If β is decreased too quickly then an excessive amount of time will be spent on the search to find (26). If it decreases too slowly, too many iterations will be necessary to obtain convergence. It seemed that initially taking $\beta = .05$ and decreasing it by 10% on each cycle worked well.

The convergence criterion used was the maximum relative constraint violation. For example, if the required coefficient of variation is γ, setting a convergence criterion of ε would mean that all CV's must be no larger than $\gamma(1+\varepsilon)$.

CONCLUSIONS

The algorithm provides an alternate program to the Agency's Huddleston, Claypool and Hocking procedure (1970) to obtain optimum solutions to multivariate problems. The motivation for the work was to provide a procedure which could be executed using an in-house micro-computer. There was no direct effort made to compare this algorithm against any other algorithms. Modest use of the program does indicate it is reasonably efficient. CPU time was 12 seconds for an optimal allocation problem (Bethel, 1985) which compares to other projects of similar size running at less than 30 seconds CPU time.

It is recommended this program be used by the Agency as an alternative to optimum sample allocation processing on Martin Marietta. This method is cost effective because it runs on an in-house machine. The method also converges quickly. Further research is suggested to evaluate the CPU run time, optimal solution, and robustness of this algorithm against the standard algorithm used by the Agency.

REFERENCES:

Avriel, M. (1976). Nonlinear Programming: Analysis and Methods. Prentice-Hall, New Jersey.

Bethel, J.W. (1985). Sample Design for the 1985 ISP/JES. SRS-USDA Staff Report No. 86.

Cochran, W.G. (1977). Sampling Techniques. John Wiley and Sons, New York.

Hartley, H.H., and Hocking, R.R. (1963). Convex Programming by Tangential Approximation. Management Science, 9, 600-612.

Huddleston, H.F., Claypool, P.L., and Hocking, R.R. (1970). Optimum Sample Allocation to Strata Using Convex Programming. App. Stat. 19, 273-278.

Kokan, A.R. (1963). Optimum Allocation in Multivariate Surveys. J. R. Statist. Soc. A, 126, 557-565.

Kokan, A.R., and Khan, S. (1967). Optimum Allocation in Multivariate Surveys: An Analytical Solution. J. R. Statist. Soc. B, 29, 115-125.

Kuhn, H.W., and Tucker, A.W. (1951). Nonlinear Programming. Proc. 2nd Berkeley Symp. Mathematical Statistics and Probability.

Williams, R.L. (1986). Sample Designs for Panel Surveys of Agricultural Production. SRS-USDA Staff Report No. 93.

APPENDIX

In this section PASCAL code for implementing the algorithm is given. The program has three subroutines. The first finds the intersection of a hyperplane with the surface S_k; the second finds the largest relative constraint violation associated with a given allocation; the third finds the cost associated with the point where S_k and a given hyperplane are tangent. The main loop first finds the hyperplane with the largest cost, then implements the routine described in section 2.

```
      program allocat (input, output);

  const
    size1 = 100;
    size2 = 50;
  type
    vector = array[1..size1] of real;
    matrix = array[1..size1] of array[1..size2] of real;
  var
        x,y,epsilon,error,beta,cost1,cost2,lambda : real;
    cnst : matrix;
    plane1,plane2,point1,point2,cost : vector;
    iteration,r,i,j,k,nstrata,nvar : integer;
    rank : array[1..size1] of integer;

(*-------------------------------------------------------------*)
(*  PROCEDURE INTERSECT takes an integer (which identifies     *)
(*  a constraint hyperplane) and returns the point of          *)
(*  intersection with the cost surface                         *)

    procedure intersect (plane : vector;
                  var inter : vector);
      var
        int : integer;
        t : real;
        w : vector;

     begin
         t := 0;
         for int := 1 to nstrata do
         begin
         w[int] := sqrt(cost[int]*plane[int]);
         t := t + w[int]
```

```
        end;
        for int := 1 to nstrata do
          if plane[int] <> 0 then inter[int] :=
                                        w[int]/(plane[int]*t)
          else inter[int] := 1.0E+10
      end;

(*-------------------------------------------------------------*)
(*  PROCEDURE "CHECK" checks to see how near a given point     *)
(*  is to the feasible region.  The global variable "error" is *)
(*  set to the largest relative constraint violation.          *)

   procedure check (vec : vector);

     var
        max,t : real;
        i, j : integer;

   begin
     max := 0;

(*  For each constraint...                                     *)

     j := 1;
     while j <= nvar do
        begin
          t := 0;

(*  ...first find a'x and 1-a'x/b, which is the error.         *)

          for i := 1 to nstrata do
             t := t + vec[i]*cnst[i,j];

          t := t-1;
          if t > max then max := t;

(*  Now max is the largest absolute constraint violation.      *)

          j := j + 1;
        end;
        error := max;
   end;

(*-------------------------------------------------------------*)
(* PROCEDURE "FINDCOST" finds the cost associated with plane p.*)

   procedure findcost (p : vector;
                var tc : real);

   var
     count : integer;

   begin
     tc := 0;
```

```
    for count := 1 to nstrata do
      tc := tc + sqrt(cost[count]*p[count]);
      tc := sqr(tc)
  end;

(*-------------------------------------------------------------*)
(* MAIN LOOP.  First the iteration variable is          *)
(* initialized, and then the data is entered.                  *)

  begin
  iteration := 0;

  writeln ('Enter number of strata, number');
  writeln ('of variables, and epsilon: ');
  read (nstrata,nvar,epsilon);

(* First the cost data for each stratum is entered.            *)

  for i := 1 to nstrata do
    read (cost[i]);

(* Now the "variance components" are read.                  *)
(* These are the a(i,j) in formula (2).              *)
(* These are read by variable, with a row for each one.    *)

  for j := 1 to nvar do
    begin
       for i := 1 to nstrata do
         read (cnst[i,j])
    end;

(* The first step is to find the hyperplane with the maximum   *)
(* value G(H) (ie., largest cost).                             *)

  for j := 1 to nvar do
    begin
    for i := 1 to nstrata do
       plane2[i] := cnst[i,j];

(*  Note that j indicates the constraint hyperplane and        *)
(*  "plane2" is the associated plane.  The first step records  *)
(*  the cost of the first plane:                               *)

    findcost (plane2,cost1);

    if j = 1 then
      begin
       r := 1;
       x := cost1
      end

(*  Now any plane with higher cost replaces the previous one.  *)

    else
```

```
            if cost1 > x then
          begin
            x := cost1;
            r := j;
          end;
      end;

(*  The following sets the initial values for the loop:        *)

    cost1 := x;
    for i := 1 to nstrata do
         plane1[i] := cnst[i,r];
    intersect (plane1,point1);
    check (point1);

(*  The iteration begins here.  The procedure is to tilt       *)
(*  plane1 toward each constraint-plane in succession, each    *)
(*  time choosing the angle which maximizes the cost           *)
(*  associated with the plane.  Beta is a parameter which      *)
(*  determines the grid of the line seach for the optimum      *)
(*  angle.  As the iterations increase, beta is decreased.     *)

       beta := 0.056;

    while error > epsilon do
     begin
    iteration := iteration + 1;
    beta := 0.9*beta;
    for j := 1 to nvar do
      begin
        x := 0;
        repeat
         y := cost1;
         x := x + beta;
         for i := 1 to nstrata do
           plane2[i] := x*cnst[i,j]+(1-x)*plane1[i];
         findcost (plane2,cost2);
         if cost2 > cost1 then cost1 := cost2;
        until cost2 <= y;
        if x > beta then
         begin
           x := x - beta;
           for i := 1 to nstrata do
                   plane1[i] := x*cnst[i,j]+(1-x)*plane1[i];
         end;
      end;
    if iteration > 25 then writeln ('Warning: iteration =',iteration);
    intersect (plane1,point1);
    check (point1);

    end;

(*  We are finished!  All that remains is to tidy things up a  *)
(*  little and then report the results.                        *)
```

```
writeln ('                    ');
writeln ('Program output: ');
writeln ('                    ');
x := 0;
   for i := 1 to nstrata do
  begin
     rank[i] := round(1/point1[i]);
     x := x + cost[i]*rank[i];
  end;
writeln ('Number of iterations =',iteration);
writeln ('The optimal cost is',cost1);
writeln ('This solution satisfies the CV constraints to within',error);
writeln ('An approximately optimal solution is the allocation');
for i := 1 to nstrata do
  write (rank[i]);
writeln;
writeln ('The cost of this solution is',x);

end.
```

☆U.S. GOVERNMENT PRINTING OFFICE: 1986-490-920/40037