R-1411-DDPAE
March 1974

Sortie Allocation by a Nonlinear Programming Model for Determining a Munitions Mix

R. J. Clasen, G. W. Graves and J. Y. Lu

A Report prepared for

DEPARTMENT OF DEFENSE
PROGRAM ANALYSIS AND EVALUATION

Rand
SANTA MONICA, CA. 90406

R-1411-DDPAE
March 1974

Sortie Allocation by a Nonlinear Programming Model for Determining a Munitions Mix

R. J. Clasen, G. W. Graves and J. Y. Lu

A Report prepared for

DEPARTMENT OF DEFENSE
PROGRAM ANALYSIS AND EVALUATION

Rand
SANTA MONICA, CA. 90406

PREFACE

For any total tactical air funding level, it is essential to maintain the balance between programmed tactical air forces and war reserve procurement for air-to-ground munitions so as to maximize the productivity of tactical sorties assigned to the air-to-surface missions.

Recently the Directorate of Defense Program Analysis and Evaluation (DDPA&E) evaluated some alternative air munitions mix methodologies. Major ones assessed were (1) the Air Force Saber Mix methodology, (2) the Navy's NAVMOR (Navy/Marine Corps Ordnance Requirements) methodology, and (3) the Rand linear program [1].

The first two methodologies involve two-step processes: selecting the "optimum" munition for each aircraft-target combination on the basis of least-cost-per-target-killed; and allocating sorties carrying the optimum munitions among targets. This sortie allocation is handled quite differently in approaches (1) and (2). Saber Mix applies the principle of marginal analysis to maximize some "military worth" objective function, and carries out the optimization for one aircraft type at a time.* The more subjective Navy methodology does not optimize any explicit stated objective function (e.g., military worth), and bases the sortie allocation on the value judgment of combat-experienced Navy officers.

The Rand approach, based on a linear programming technique, considers sortie allocation, target selection, and weapon selection as all part of the same optimization process in which munition procurement costs are minimized. Unlike Saber Mix, it considers sorties from all aircraft types simultaneously in the optimization.

Upon examining these alternative approaches, DDPA&E concluded that it is desirable to develop a methodology based on a mathematical programming approach for maximizing some military worth objective function. The problem was formulated as a nonlinear programming model

---

*The Air Force is currently modifying Saber Mix so that all aircraft are allocated at once.

with given constraints and objective form, and Rand was requested to develop an algorithm and an associated computer code for solving the problem. The algorithm and computer code presented in this report are intended for use by the DPA&E's own staff. This methodology synthesizes the best features of both Air Force and Navy methods, and could serve as a common sortie allocation methodology for establishing both Service's air-to-surface munitions requirements.

Although the computer program was developed for sortie allocation problems, the nonlinear programming algorithm obviously has applicability to a wide range of problems. Hence, it is discussed in some detail and in more general terms than the sortie allocation problem calls for.

SUMMARY

This report presents a mathematical programming approach to maximize a military worth objective function subject to resource constraints. The formulation takes account of the diminishing returns obtained with incremental capability increases, thus producing a problem of the convex programming type.

A very general nonlinear programming algorithm is presented, along with a discussion of its numerical properties and a proof of its convergence for the convex programming problem. This is followed by a description of the computer input for the specific problem studied, and by an example problem with its associated computer output.

An appendix details the use of the general algorithm for applications that differ from the one considered here. This usage involves altering several PL/1 procedures to the form desired.

CONTENTS

I. INTRODUCTION

This report describes a nonlinear programming algorithm and its associated computer code written in the PL/1 programming language. The nonlinear programming techniques are used to select an air-to-surface munitions mix to be stockpiled as war reserve inventory. The specific problem formulated allocates tactical sorties to different target types so as to maximize some "military worth" objective function. The solution, combined with data regarding the type and amount of munitions to be carried in each sortie, provides the basis for determining the quantity and quality of the air munitions to be procured for the stockpile. To clarify where the algorithm fits in the total process of munitions mix selection, we first discuss our general approach to the problem.

The proposed munitions mix selection methodology as applied to a given interval consists of three steps: (1) it selects a munition for each aircraft-target combination based on a criterion of least-cost-per-expected-target-killed; (2) it allocates sorties from all aircraft types--each sortie carrying the selected "optimum" munition--among target types; and (3) it computes the required munition mix based on the first two steps. The diagram in Fig. 1 displays the logical relationships between these three steps and various input factors.

In allocating sorties among targets, we assume that a law of diminishing marginal productivity can be in effect, i.e., the number of targets killed per sortie decreases for each successive sortie. The degree to which the law of diminishing marginal productivity is in effect is determined by a single parameter for each target type. There are several possible justifications for considering diminishing marginal productivity.

First, there is a question of target availability. As more sorties are launched and more targets killed, the difficulty of finding live targets increases. Second, remaining targets would become harder to kill because they would either offer a stiffer resistance or be dispersed if they are mobile. Third, attacking forces may become less

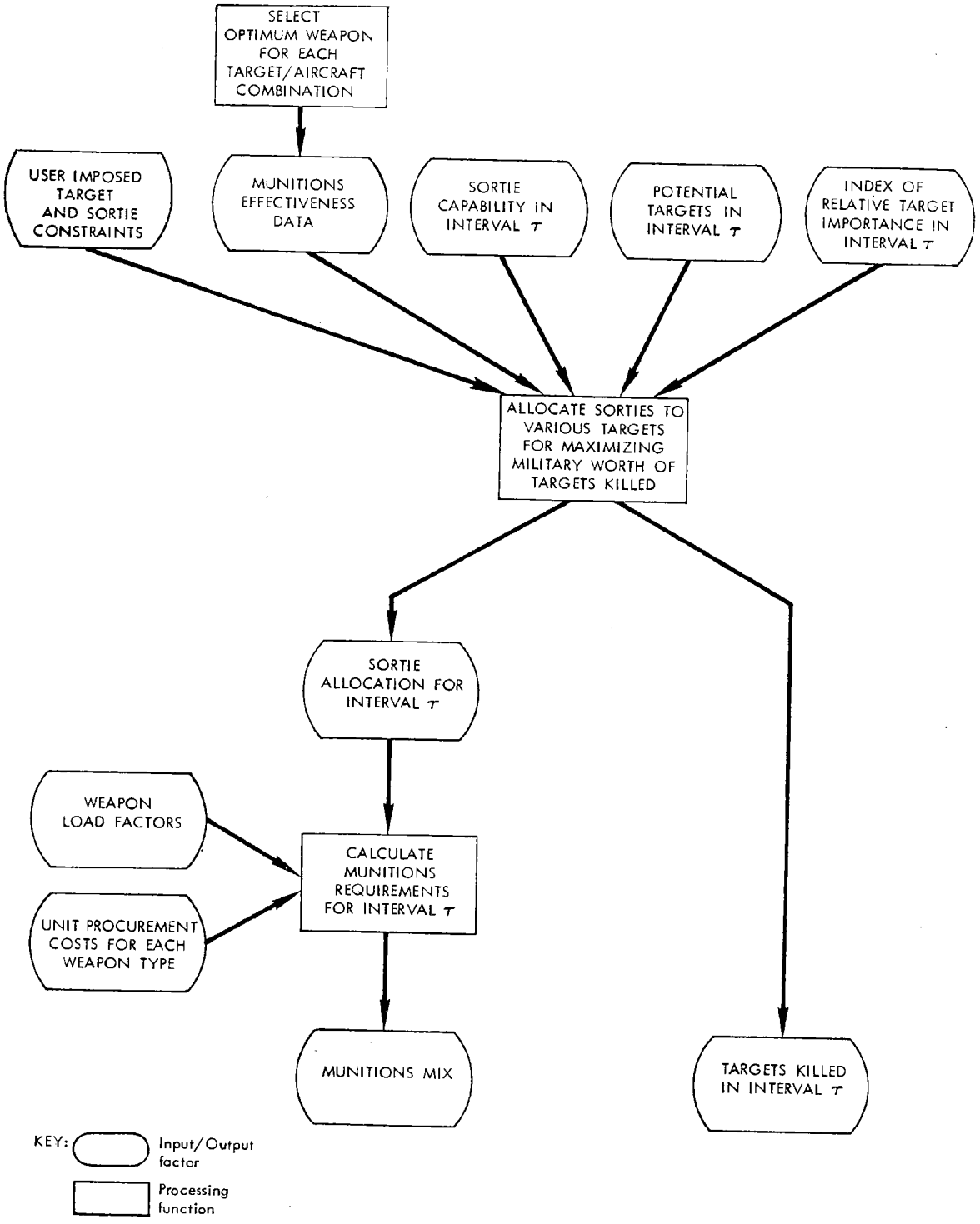


Fig. 1 — Input-Output relationship in munitions mix calculation process

effective for a host of reasons: for example, attrition may have sapped their strength, or tactics have become known to the enemy and elements of surprise no longer exist.

This methodology would provide an analytical mechanism and a reproducible linkage between the threat (potential target arrays), counterforces (available sorties), strategy/tactics (with implications for relative target values), and munition selection. This would allow a planner to examine, using a consistent logical framework, the different mix of weapons demanded when the relevant variables and assumptions change.

As stated above, the primary purpose of the proposed methodology is to establish war reserve munitions requirements that are consistent with planning assumptions concerning programmed sortie capability, potential enemy threat, and the effectiveness of current as well as future tactical air munitions. In addition, the methodology has applications to other policy issues. Two examples follow.

1. *Tactical Air Sortie Capability versus Air Munitions Procurement.* The productivity of tactical air forces, measured in terms of target kill capability, depends on the number of sorties that the forces can generate and the quantity and quality of munitions that they can deliver to targets. Within any total tactical air funding level, force productivity can be changed by varying the resource allocation between sortie capability and munitions. If all the resources were spent on aircraft, the tactical air forces would obviously be less effective because they would lack the needed quantity as well as quality of munitions. Hence, it is desirable to strive for a balance between programmed tactical forces and air munitions war reserve procurement. This issue is becoming more critical because fiscal constraints for future military procurement are expected to be more stringent.

   The proposed methodology is primarily designed for determining the most effective munitions mix for a *given level* of sortie capability. By making a series of parametric studies

in which the sortie capability level is varied, it is possible to determine the corresponding "optimum" munitions stockpile and gain insight into the nature of tradeoffs between sortie capability and munitions.

2. *Relative Effectiveness of Munitions*. The effectiveness of munitions in development can be done in the context of a total tactical warfare environment in which a scenario is represented by available sorties and opposing enemy targets and allows the planner to determine how the weapon being developed will compete with other weapon types already in production. Analysis results will help determine the contributions that new munitions will make to the productivity of the tactical air forces.

The remainder of this report formulates the sortie allocation problem as a nonlinear programming model, outlines the algorithm, and describes the computer code. The appendixes present technical details of the algorithm and the code.

II. MATHEMATICAL STATEMENT OF THE PROBLEM

1. PROBLEM FORMULATION

The problem is to determine the number of sorties S_{ij}, where i is the aircraft type and j is the target type, by maximizing an objective function subject to given constraints. The following notation is used:

- i = aircraft type index
- I = number of aircraft types
- j = target type index
- J = number of target types
- T_j = number of type j targets
- D_j = number of dead targets
- P_{ij} = number of type j targets killed by a type i sortie.
- S_{ij} = number of sorties, defined above; the independent variables.
- C_j = target parameter, between 0 and 1, designating for each target type the degree to which its status (dead or alive) can be determined. This parameter controls the extent to which the law of diminishing productivity applies.
- S_i = upper limit on number of sorties for aircraft type i.
- V_j = value (or value paraters) of target j.
- ℓ_j = lower bound on targets killed.

Other than S_{ij}, all of the above subscripted variables are given constants for a specific problem. K_j, defined by the following expression,* is the number of kills of target type j:

$$K_j = \frac{T_j}{C_j}\left\{1 - \exp\left[-\frac{C_j}{T_j}\left(\alpha_j + \sum_{i=1}^{I} P_{ij}S_{ij}\right)\right]\right\}, \qquad (1.1)$$

where α_j is the constant

*DDPA&E suggested this function to us. It is similar to the objective function of the Saber Mix methodology.

$$\alpha_j = -\frac{T_j}{C_j} \log\left(1 - \frac{C_j}{T_j} D_j\right). \tag{1.2}$$

The problem is to determine nonnegative variables

$$(S_{ij}: \quad i = 1, 2, \ldots, I, \quad j = 1, 2, \ldots, J)$$

that satisfy the *sortie constraints*

$$\sum_{j=1}^{J} S_{ij} = S_i, \quad i = 1, 2, \ldots, I; \tag{1.3}$$

the *target constraints*

$$\ell_j \le K_j \le T_j, \quad i = 1, 2, \ldots, J; \tag{1.4}$$

and the side constraints each having the general form

$$\sum_{j=1}^{J} a_j S_{ij} \le 0$$

for some aircraft type i, in order to maximize

$$\sum_{j=1}^{J} V_j (K_j - D_j). \tag{1.6}$$

The sortie constraints have the form of an unscaled generalized upper bound. This form permits the upper bound

$$S_{ij} \le S_i \quad \text{for all } i, j \tag{1.7}$$

to be placed on each variable. (Note that S_{ij} are the variables, S_i are constants). Each variable has a lower bound of zero. These upper bounds do not actually constrain the solution because they are redundant to the sortie constraints.

The target constraints appear to be nonlinear; however, this

nonlinearity can be eliminated by solving the inequality for the linear term

$$\sum_{i=1}^{I} P_{ij}S_{ij}.$$

Then the upper inequality of the target constraints takes the form

$$\sum_{i=1}^{I} P_{ij}S_{ij} \leq -\frac{T_j}{C_j} \log (1 - C_j) - \alpha_j \qquad (1.8)$$

for $j = 1, 2, \ldots, J$. When C_j is unity, the right-hand side of the inequality is infinite, so that targets for which C_j is unity will not require an upper target constraint. Similarly, the lower inequality of the target constraints takes the form

$$\sum_{i=1}^{I} P_{ij}S_{ij}(t) \geq -\frac{T_j}{C_j} \log \left(1 - \frac{C_j}{T_j}\ell_j\right) - \alpha_j , \qquad (1.9)$$

which is not required if $\ell_j \leq D_j$ because $\alpha_j \geq D_j$ follows from a Taylor expansion of 3.1, given below under "Numerical Approximations."

The side constraints are of two types; upper-bound sortie constraints and lower-bound sortie constraints. The upper-bound constraints limit the sorties by a specific aircraft type against a specified list of target types to a fraction of the total number of sorties flown by that aircraft type. Thus if $\mathcal{J}$ is a subset of $\{1, 2, \ldots, J\}$, then an upper-bound constraint for an aircraft i has the form

$$\sum_{\mathcal{J}\in J} S_{ij} \leq f \sum_{j=1}^{J} S_{ij}, \qquad (1.10)$$

where $0 < f < 1$. The lower-bound constraints have the same form with the reverse inequality.

The parameters of the problem, T_j, D_j, ℓ_j, C_j, V_j, and P_{ij}, are all nonnegative with $0 \leq C_j \leq 1$ and $D_j \leq T_j$. T_j must be strictly positive so that division by zero will not be attempted. Because the

logarithm of 1 + x is subject to large round-off errors when x is small, an approximating formula is used. This modification is discussed below under "Numerical Approximations."

2. CONVEXITY OF THE OBJECTIVE FUNCTION

The problem as stated has linear constraints, and is also bounded in its feasible region. To obtain the solution to this type of nonlinear programming problem readily, the objective function must be convex in a minimization problem. Let the objective function be denoted by ϕ:

$$\phi = \sum_{j=1}^{J} V_j(K_j - D_j). \tag{2.1}$$

Then

$$\frac{\partial \phi}{\partial S_{ij}} = V_j \frac{\partial K_j}{\partial S_{ij}}, \tag{2.2}$$

because K_j depends only on S_{ij} variables of the same target type. Thus

$$\frac{\partial^2 \phi}{\partial S_{ij} \partial S_{\bar{i}\bar{j}}} = \delta_{j\bar{j}} \frac{\partial^2 K_j}{\partial S_{ij} \partial S_{\bar{i}j}} V_j, \tag{2.3}$$

where $\delta_{j\bar{j}}$ is one if $j = \bar{j}$, and zero otherwise. The second partials on the right can be evaluated

$$\frac{\partial^2 K_j}{\partial S_{ij} \partial S_{\bar{i}j}} = - P_{ij} P_{\bar{i}j} \frac{C_j}{T_j} \exp\left[- \frac{C_j}{T_j} \left(\alpha_j + \sum_{k=1}^{I} P_{kj} S_{kj} \right) \right],$$

so that in general

$$\frac{\partial^2 \phi}{\partial S_{ij} \partial S_{\bar{i}\bar{j}}} = - \delta_{j\bar{j}} V_j P_{ij} P_{\bar{i}j} \frac{C_j}{T_j} \exp\left[- \frac{C_j}{T_j} \left(\alpha_j + \sum_{k=1}^{I} P_{kj} S_{kj} \right) \right].$$

If the matrix of second partial derivatives (called the Hessian) of the objective function is nonnegative semidefinite at each feasible

point, then we consider the objective function convex on its feasible domain. The matrix developed here is the Hessian, H, of the negative of the objective function (that is minimized); H is nonpositive semidefinite if for any vector π the scalar $\pi^{\tau}H\pi$ is nonpositive.

Theorem: $\dfrac{\partial^2 \phi}{\partial S_{ij} \partial S_{\bar{i}\bar{j}}}$ *is nonpositive semidefinite.*

Proof: The Hessian assumes the form of a block diagonal matrix; it is sufficient to show that each block (with fixed j) is nonpositive semidefinite. If V_j is zero, the j*th* block is all zeros and thus trivially semidefinite: assume therefore that V_j is positive. It is then only a scale factor and can be ignored in any calculations. Thus it is sufficient to show that the submatrices

$$\frac{\partial^2 K_j}{\partial S_{ij} \partial S_{\bar{i}j}} \qquad \begin{matrix} i = 1, 2, \ldots, I \\ \bar{i} = 1, 2, \ldots, I \end{matrix}$$

for j = 1, 2, ..., J are nonpositive semidefinite for each j. Choose any vector $\pi = (\pi_1, \pi_2, \ldots, \pi_I)^{\tau}$, and form the double sum

$$\sum_{i=1}^{I} \sum_{\bar{i}=1}^{I} \pi_i \pi_{\bar{i}} \frac{\partial^2 K_j}{\partial S_{ij} \partial S_{\bar{i}j}}$$

$$= \left(- \sum_{i=1}^{I} \sum_{t=1}^{I} \pi_i \pi_{\bar{i}} P_{ij} P_{\bar{i}j} \right) \frac{C_j}{T_j} \exp \left[- \frac{C_j}{T_j} \left(\alpha_j + \sum_{k=1}^{I} P_{kj} S_{kj} \right) \right]$$

$$= - \exp \left[- \frac{C_j}{T_j} \left(\alpha_j + \sum_{k=1}^{I} P_{kj} S_{kj} \right) \right] \cdot \frac{C_j}{T_j} \left[\sum_{i=1}^{I} P_{ij} \pi_i \right]^2 .$$

Since this expression is nonpositive for any choice of π, the matrix is nonpositive semidefinite. QED.

This discussion shows that we are dealing with a convex programming problem here. Such problems have the property that any local optimum is a global optimum. This property allows the use of gradient techniques to solve the problem.

3. NUMERICAL APPROXIMATIONS

As mentioned above, because of round-off errors certain functions may be subject to large errors, even though long precision arithmetic is used. Let us first define the function $f_1(x)$ as follows:

$$f_1(x) = \begin{cases} \frac{1}{x} \log (1 + x) & \text{for } x \neq 0 \; x > -1. \\ 1 & \text{for } x = 0 \\ \text{undefined} & \text{for } x \leq -1. \end{cases}$$

Note that since

$$\lim_{x \to 0} \frac{1}{x} \log (1 + x) = 1,$$

$f_1(x)$ is continuous for $x > -1$. Its derivative is (never used computationally)

$$f_1'(x) = \begin{cases} \frac{1}{x^2} \left[\frac{x}{1 + x} - \log (1 + x) \right] & \text{for } x \neq 0 \text{ and } x > -1 \\ - 1/2 & \text{for } x = 0. \end{cases}$$

$f_1'(x)$ is also continuous for $x > -1$. If the occurrences of $\log (1 + x)$ above are replaced by $f_1(x)$, the results are the following: Equation (1.2) becomes

$$\alpha_j = D_j f_1 \left(- \frac{C_j D_j}{T_j} \right); \qquad (3.1)$$

Eq. (1.8) becomes

$$\sum_{i=1}^{I} P_{ij} S_{ij} \leq T_j f_1(-C_j) - \alpha_j; \tag{3.2}$$

and Eq. (1.9) becomes

$$\sum_{i=1}^{I} P_{ij} S_{ij} \geq \ell_j f_1\left(-\frac{C_j \ell_J}{T_j}\right) - \alpha_j. \tag{3.3}$$

In practice, the second-order Taylor expansion for log (1 + x) is used when x is small; the computer program defines

$$f_1(x) = \begin{cases} \frac{1}{x} \log (1 + x) & \text{if } x > -1 \quad \text{and } |x| > 10^{-4} \\ 1 - \frac{1}{2} x & \text{if } |x| \leq 10^{-4}. \end{cases}$$

This further implies that although C_j appears in the denominator of Eqs. (1.2), (1.8), and (1.9), because (3.1) through (3.3) are used in their place, a *value of zero is legal* for C_j. When $C_j = 0$, Eqs. (3.1) through (3.3) become

$$\ell_j \leq \sum_{i=1}^{I} P_{ij} S_{ij} + D_j \leq T_j, \tag{3.4}$$

which is, in fact, the desired linear case.

The second function for which an approximation is used is

$$f_2(x) = \begin{cases} \frac{1 - e^{-x}}{x} & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases}.$$

As with f_1, f_2 is continuously differentiable. The implemented second-order approximation is

$$f_2(x) = \begin{cases} \dfrac{1 - e^{-x}}{x} & \text{if } |x| > 10^{-4} \\ 1 - \dfrac{x}{2} & \text{if } |x| \leq 10^{-4} \end{cases}.$$

This function is used to compute K_j (Eq. 1.1) by

$$K_j = \tilde{P}_j f_2\left(\frac{C_j \tilde{P}_j}{T_j}\right), \tag{3.5}$$

where

$$\tilde{P}_j = \alpha_j + \sum_{i=1}^{I} P_{ij} S_{ij} . \tag{3.6}$$

Since $f_2(0) = 1$, when $C_j = 0$, $K_j = \tilde{P}_j$, and $\alpha_j = D_j$, so that if $C_j = 0$, then

$$K_j = D_j + \sum_{i=1}^{I} P_{ij} S_{ij};$$

again the linear case.

III. THE NONLINEAR PROGRAMMING ALGORITHM

1. INTRODUCTION

The basic technique for solving a general nonlinear programming algorithm via a sequence of local linear programming problems [2] can be rigorously demonstrated to converge in a finite number of steps to a local linear stationary point. The local linear stationary points depending on the conditions imposed on the original problems are interpreted as global optimal points, local optimal points, saddlepoints, or points demonstrating infeasibility of the original nonlinear system. As is generally the case, however, this rigorous demonstration of convergence sheds little light on the rapidity of convergence for practical problems and hence its utility as a practical test for real problems. Fortunately, the algorithm developed here has been extensively used on a wide variety of real problems for over ten years [3,4,5,6]. It has evolved in response to actual problem needs and has performed well. It represents a general procedure in that it handles general nonlinear problems, but it is also efficient for the class of problems described in Sec. II (i.e., problems with linear constraints). Other general methods with some computational success are described in [7,8,9,10].

2. OUTLINE OF THE NONLINEAR PROGRAMMING ALGORITHM

The algorithm employed for the nonlinear programming problems of this report is a general purpose algorithm that solves problems of the form

Subject to

$$g^i(y) \leq 0 \quad i = 1, \ldots, m-1$$

$$\underset{y}{\text{Min}} \quad g^m(y), \tag{2.1}$$

where y is a vector in E^n, and $g^i(y)$ $(i = 1, \ldots, m)$ are differentiable functions. It is a "local," "gradient," "stepwise" correction descent

algorithm (Fig. 2). By a "stepwise" procedure we mean that given a point y^o in the domain of the functions, a "correction" vector Δy is determined and a new point $y = y^o + k\,\Delta y$ is used for the successor "step". It is a "local" method because the correction direction Δy and its length (determined by the scalar k) are obtained from the behavior of the system in a "sufficiently" small neighborhood of the current point y^o. It is a "gradient" technique inasmuch as the gradients of the function $g^i(y)$ are principally used to obtain the correction direction. The algorithm is a "local-gradient" technique because of reliance on the approximation of the functions $g^i(y)$ by the approximation theorem.

Theorem 1. Approximation

Let $g \in C'$ *in an open region* D, *and let* E *be a closed bounded subset of* D. *Let*

$$\nabla g(y^o) = \left[\frac{\partial g(y^o)}{\partial y_1}, \frac{\partial g(y^o)}{\partial y_2}, \ldots, \frac{\partial g(y^o)}{\partial y_n} \right] .$$

be the "gradient" of g at the point $y^o \in E$:
If

$$g(y^o + \Delta y) = g(y^o) + \nabla g(y^o)\Delta y + R(y^o, \Delta y) , \qquad (2.2)$$

then

$$\lim_{\Delta y \to 0} \frac{R(y^o, \Delta y)}{|\Delta y|} = 0$$

uniformly for $y^o \in E$.

The proof of this result is found in Buck.*

The direction of improvement Δy in the stepwise procedure is obtained from the function approximation in (2.2) by estimating the $R^i(y^o, \Delta y)$ and solving the associated local linear programming problem:

*See theorem 8 on page 184 of Ref. 11.

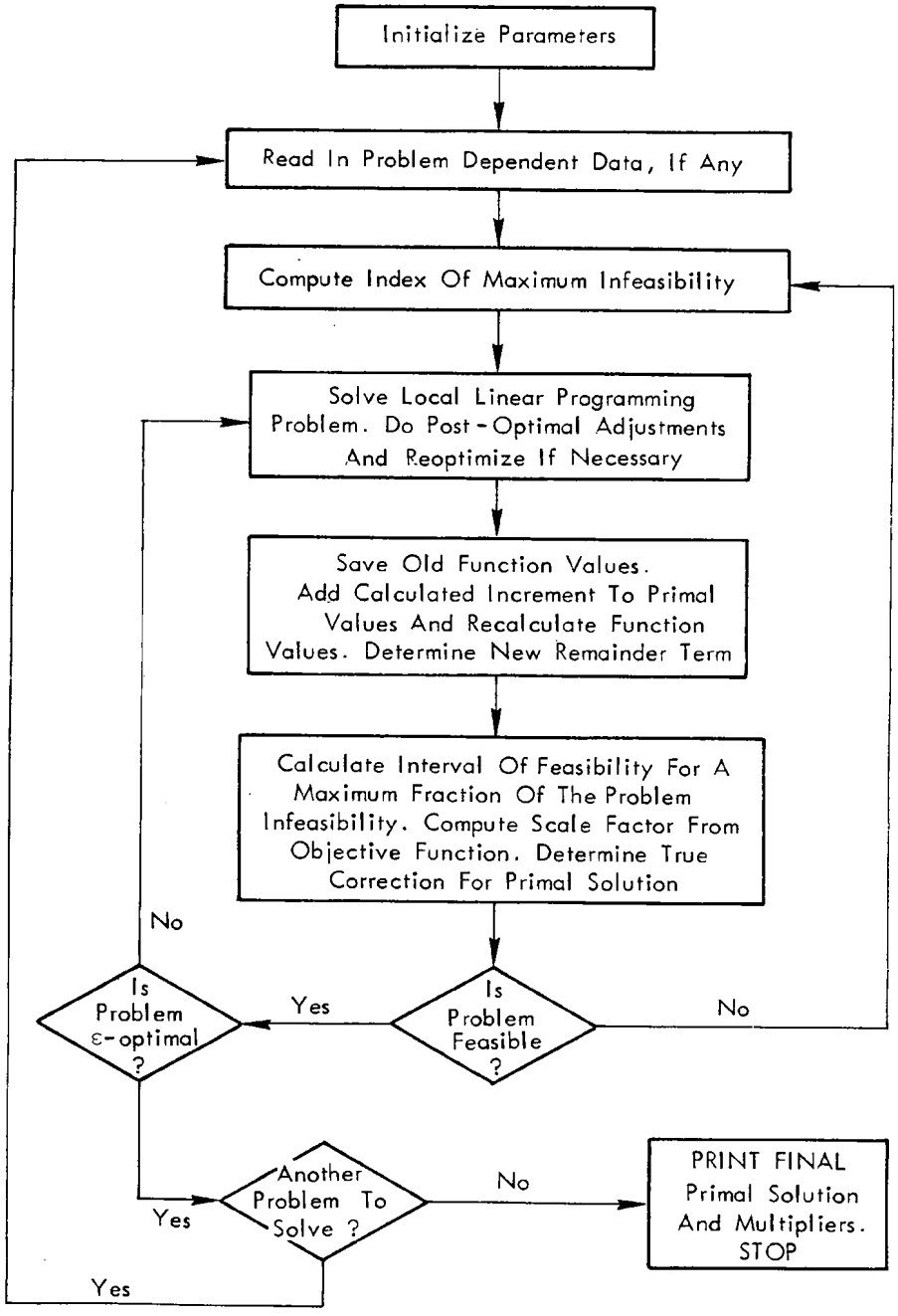


Fig. 2 — General flow of nonlinear algorithm

Subject to the constraints

$$\nabla g^i(y^o)\Delta y \le -g^i(y^o) - \bar{k}\, r^i \qquad i = 1, 2, \ldots, m-1 , \tag{2.3}$$

choose Δy to minimize $\nabla g^m(y^o)\Delta y$.

The r^i are an arbitrary set of positive constants that serve as surrogates for the $R^i(y^o, \Delta y)$, where $r^i = 0$ for the strictly linear functions. Of course the $R^i(y^o, \Delta y)$ are truly functions of the direction Δy being sought, and substituting positive constants causes the solution of the local linear programming problem to be an approximation. In practice, the r^i are taken to be the $R^i(y^o, \Delta y)$ obtained from the previous Δy correction vector. The $\bar{k}$ is used to adjust the solutions of the local linear programming problems parametrically. In general, as $\bar{k}$ decreases it becomes easier to achieve feasibility in the local linear programs, but unfortunately the gain that can be assured in the convergence of the nonlinear problem also decreases. As will be shown in the convergence proof, it is necessary to have $\bar{k} \ge s > 0$ for some fixed s to prevent convergence to a nonoptimal limit point.

3. NATURE OF THE SOLUTION

To discuss the convergence of the algorithm we must develop the concept of an ε-solution of the problem. Given any $\varepsilon > 0$, it is first necessary to construct a suitable "feasible" set. This consists of exhibiting an ε such that $0 < \varepsilon_1 \le \varepsilon$, and defining the ε-feasible set $F = \left\{ y \mid y \in E \text{ and } g^i(y) \le \varepsilon_1,\ i = 1, \ldots, m-1 \right\}$. The solutions y^* are then characterized as

Global ε-minimums

For all $y \varepsilon F$

$$g^m(y^*) \le g^m(y) + \varepsilon \tag{3.1}$$

or

Local ε-minimums

For some $\delta > 0$ and $y \varepsilon F$ such that

$$|y - y^*| < \delta$$

$$g^m(y^*) \leq g(y) + \varepsilon \ . \tag{3.2}$$

Obviously, the concept of an ε-solution is necessary when treating real valued functions. The solutions in general will be real valued numbers and an answer can be determined only when the level of accuracy desired is specified. A problem is considered completely solved when for an arbitrary positive ε an ε-solution can be guaranteed in a finite number of steps.

4. REVIEW OF LINEAR PROGRAMMING RESULTS

We also require the following well-known fundamental results from the theory of linear programming. A general mixed linear programming problem can always be stated in both a primal and dual form.

Primal Form

Subject to the constraints

$$\begin{aligned} &\sum_q a_{iq} y_q = a_{in}; \quad 1 \leq i \leq m_1 \\ &\sum_q a_{iq} y_q \leq a_{in}; \quad m_1 < i < m \\ &y_q \geq 0 \quad n_1 < q < n \ , \end{aligned} \tag{4.1}$$

minimize

$$\sum_q a_{mq} y_q \ .$$

Dual Form

Subject to the constraints

$$\sum_p x_p a_{pj} = a_{mj}; \quad 1 \le j \le n_1$$

$$\sum_p x_p a_{pj} \le a_{mj}; \quad n_1 < j < n \tag{4.2}$$

$$x_p \le 0 \quad m_1 < p < m ,$$

maximize

$$\sum_p x_p a_{pn} .$$

The fundamental inequality of linear programming states:

Given any feasible solution y_q of the constraints in (4.1) and any feasible solution x_p of the constraints in (4.2), then

$$\sum_q a_{mq} y_q \ge \sum_p x_p a_{pn} . \tag{4.3}$$

The solvability or unsolvability of the linear programming problem as well as the relationship between the primal and dual problems can be summarized in a general theorem.

Theorem 2. Linear Programming

Given a linear programming problem, exactly one of the following conditions holds:

1. *There exists a finite value* v *and feasible vectors* y_q^* *and* x_p^* *of* (4.1) *and* (4.2) *such that*

$$\sum_q a_{mq} y_q^* = \sum_p x_p^* a_{pn} = v \tag{4.4}$$

(and by the fundamental inequality (4.3), they must then be optimal feasible vectors).

2. *The constraints for the primal problem are inconsistent and either the constraints of the dual problem are inconsistent or the dual extremal function is unbounded.*
3. *The primal extremal function is unbounded and the constraints for the dual problem are inconsistent.*

For a proof of this result, see a standard linear programming text such as [12]. Note that the dual of Eq. (2.3) is the following:

Subject to the constraints

$$\sum_{p=1}^{m-1} x^p \nabla g^p(y^o) = \nabla g^m(y^o) \ , \tag{4.5}$$

find $x^p \leq 0$ to maximize

$$\sum_{p=1}^{m-1} [-x_p][g^p(y^o) + \bar{k}r^p] \ .$$

5. DEFINITION OF BOUNDS USED IN THE THEORETICAL DEVELOPMENT

There are three bounds required in our development.

Bound 1

$$\sum_{p} (-x_p) \leq B_1 \ . \tag{5.1}$$

The bound on the magnitude of the dual variables encountered in solving the local linear programming problems has to be imposed as an additional condition on the problem. This condition merely acknowledges a limit to our computational ability to distinguish nonsingularity in inverting the basis associated with the local linear programming problem. It is sufficient to satisfy a Kuhn-Tucker constraint qualification.*

*See page 39 of Ref. 13.

Bound 2

$$|\Delta y| \leq B_2 \ . \tag{5.2}$$

Since E is a bounded set, the vectors y of interest are bounded. By the triangle inequality

$$|\Delta y| = |y - y^o| \leq |y| + |y^o| \ ,$$

and the correction vector is also bounded. Therefore, we can always superimpose constraints of the form $L^i \leq y_i \leq U^i$, which can be treated algebraically without explicit introduction into the constraint set. These bounds will always exclude obtaining an unbounded solution in solving the local linear programming problems (alternative 3 of theorem 2).

Bound 3

$$|R^i(y^o, \Delta y)| \leq B_3, \quad \{i = 1, \ldots, m\} \ . \tag{5.3}$$

Transposing (2.2),

$$R^i(y^o, \Delta y) = g^i(y) - g^i(y^o) - \nabla g^i(y^o)\Delta y \ .$$

But since $g^i(y) \varepsilon C'$, both $g^i(y)$ and $\nabla g^i(y^o)\Delta y$ are continuous functions, and the difference of continuous functions is a continuous function. Hence $R^i(y^o, \Delta y) = p(y, y^o)$ is a continuous function over a compact set ExE and is therefore bounded. Taking the largest bound over the finite set of bounds for the $R^i(y^o, \Delta y)$, we obtain an overall bound for the remainder terms.

6. INCONSISTENCY

To define a local linear stationary point of our algorithm and to demonstrate convergence in a finite number of steps, we must first

establish a crucial result on the consistency of the local linear programming problems.

Let $M = \{1, 2, \ldots, m - 1\}$ be the indices of the constraints. Let a subset of the indices of the constraints be denoted by H, and let w be an element of M. (In the following result w and H can be shown).

A local linear programming (L.L.P.) problem is consistent if there is a Δy that satisfies all of the constraints. It is assumed that the range restrictions that bound the problem are numbered among the constraints. Similarly, an L.L.P. is consistent with respect to $H \subseteq M$ if there is a Δy that satisfies the constraints with indices in H. Then we have

Lemma 1.

Either the L.L.P. problems are consistent at y^o *for some* $\bar{k} \geq \varepsilon/(B_3 + B_1B_3)$ *or there exists an index* w, *a set* H, *and a set of multipliers* x_p, *where* $x_p \leq 0$ *such that*

$$\sum_{p \in H} (-x_p) g^p(y^o) > -g^w(y^o) - \varepsilon \ .$$

Proof

Let $H = \left\{ i \mid \nabla g^i(y^o)\Delta y \leq \left[-g^i(y^o) - \bar{k}r^i\right] \text{ for some } \Delta y \right\}$. H is a consistent subset of M. Such a subset always exists since one can always choose $\Delta y = L_i - y^o$ and the set H will contain the range restriction constraints of the form $L_i \leq y_i^o + \Delta y_i \leq U_i$. Let w be an index such that

$$\nabla g^w(y^o)\Delta y > -g^w(y^o) - \bar{k}r^w \ ,$$

so that constraint w is infeasible at Δy with the current choice of $\bar{k}$. Solve the subproblem:

Subject to

$$\nabla g^i(y^o)\Delta y \leq g^i(y^o) - \bar{k}r^i \quad i\epsilon H \ ,$$

minimize

$$\nabla g^w(y^o)\Delta y \ . \tag{6.1}$$

This problem is consistent and bounded by construction. Therefore, alternative 1 of Theorem 2 occurs and we have a w and dual variables $x_p \leq 0$ such that

$$\nabla g^w(y^o) = \sum_{p\epsilon H} x_p \nabla g^p(y^o)$$

and

$$\nabla g^w(y^o)\Delta y = \sum_{p\epsilon H} x_p[-g^p(y^o) - \bar{k}r^p] \ .$$

Let $B = \{i \mid i\epsilon H$ and i indexes the basic constraints of (6.1)$\}$. Maintaining this basis, the constraint w will be consistent for any $\bar{k}$ such that

$$\nabla g^w(y^o)\Delta y = \sum_{p\epsilon H} x_p[-g^p(y^o) - \bar{k}r^p] \leq -g^w(y^o) - \bar{k}r \ ,$$

or solving for $\bar{k}$ such that

$$\bar{k} \leq -\left[g^w(y^o) + \sum_{p\epsilon H} (-x_p)g^p(y^o)\right]\left[r^w + \sum_{p\epsilon H} (-x_p)r^p\right]. \tag{6.2}$$

Now starting with any $\bar{k}$, it can be adjusted (down) to remove any inconsistencies generated. And since the basis B and the associated w can never repeat (because $\bar{k}$ is monotone) in a finite number of steps, we reach consistency in the L.L.P., or otherwise $\bar{k}$ will be chosen such that

$$\frac{\varepsilon}{(B_3 + B_1 B_3)} > \bar{k} = - \frac{\left[g^w(y^o) + \sum_{p \in H} (-x_p) - g^p(y^o) \right]}{\left[r^w + \sum_{p \in H} (-x_p) r^p \right]}$$

or

$$\frac{\left[r^w + \sum_{p \in H} (-x_p) r^p \right]}{(B_3 + B_1 B_3)} \varepsilon > -\left[g^w(y^o) + \sum_{p \in H} (-x_p) g^p(y^o) \right] ;$$

or since

$$\frac{\left[r^w + \sum_{p \in H} (-x_p) r^p \right]}{(B_3 \pm B_1 B_3)} < 1 ,$$

then

$$\varepsilon > -\left[g^w(y^o) + \sum_{p \in H} (-x_p) g^p(y^o) \right] ,$$

or

$$\sum_{p \in H} (-x_p) g^p(y^o) > -g^w(y^o) - \varepsilon . \quad \text{QED.}$$

7. CONVERGENCE OF THE ALGORITHM

We can now give the conditions for a stationary point of the algorithm.

Definition: A point y^o is a *Local Linear Stationary Point* of the nonlinear problem (2.1) if the dual variables of the L.L.P. at y^o satisfy one of the following conditions:

1) $\sum_{p} (-x_p) > B_1$ (insufficient resolution)

2) There exists a w and H such that

$$\sum_{p\in H} (-x_p) g^p(y^o) \geq -g^w(y^o) - \varepsilon \quad \text{(inconsistency)}$$

3) $$\sum_{i=1}^{m-1} (-x_i) g^i(y^o) \geq -\varepsilon \quad \text{(optimality)} .$$

We provide an interpretation of the consequences of terminating the algorithm with these stopping criteria after demonstrating that they are the only conclusions.

Theorem 3. Convergence

Let $g^1, g^2, \ldots, g^m$ be convex functions. Then it is always possible to choose $\bar{k}$ and k in such a manner that in a finite number of steps a local linear stationary point is reached.

Proof:

Choose $\bar{k} = \varepsilon/(B_3 + B_1 B_3)$.

To establish k, we must dominate the remainder terms $R^i(y^o, \Delta y)$ and require the following consequences of the approximation theorem (1):

For any $y^o \in E$, there exists a $\delta > 0$ such that for $k \cdot |\Delta y| \leq \delta$,

$$\frac{R^i(y^o, k\Delta y)}{k|\Delta y|} < \frac{\bar{k} r^i}{B_2} , \quad i = 1, \ldots, m-1 , \tag{7.1}$$

$$\frac{R^m(y^o, k\Delta y)}{k|\Delta y|} < \frac{\varepsilon}{B_2} \frac{1 \cdot}{2(1 + B_1)} . \tag{7.2}$$

Recalling that $|\Delta y| \leq B_2$, and now choosing $k = \delta/B_2$, we have

$$\frac{R^i(y^o, k\Delta y)}{k} < \bar{k} r^i , \quad i = 1, \ldots, m-1 , \tag{7.3}$$

$$R^m(y^o, k\Delta y) < \frac{\varepsilon \cdot \delta}{B_2} \frac{1}{2(1 + B_1)} . \tag{7.4}$$

Let

$$g_{max} = \max_i \left[g^i(y^o) \,|\, g^i(y^o) > 0 \right]$$

or

$$g_{max} = 0 \quad \text{if } g^i(y^o) \le 0 \quad i = 1, \ldots, m-1 . \tag{7.5}$$

The feasibility of the constraints is maintained or the infeasibility is decreased (as measured by g_{max}) when k is chosen to satisfy the constraints,

$$g^i(y) = g^i(y^o + k\Delta y) = g^i(y^o) + \nabla g^i(y^o)\Delta y \cdot k + R^i(y^o, k\Delta y) \le d \cdot g_{max} \tag{7.6}$$

with $0 \le d < 1$, and as close to zero as possible when $g_{max} > 0$. At the conclusion of the local linear programming problem we are consistent or by Lemma 1 we satisfy the inconsistency stationary condition 2 and terminate. When we are consistent, we have from (2.3) the feasibility conditions. Thus,

$$\nabla g^i(y^o)\Delta y \cdot k \le [-g^i(y^o) - \bar{k}r^i]k, \quad i = 1, 2, \ldots, m-1 . \tag{7.7}$$

Substituting this into (7.6), it is sufficient to choose k such that the stronger condition holds:

$$g^i(y^o) + [-g^i(y^o) - \bar{k}r^i]k + R^i(k\Delta y) \le d \cdot g_{max} \tag{7.8}$$

or

$$-\bar{k}r^i \le \frac{d\, g_{max} - (1-k)g^i(y^o)}{k} - \frac{R^i(y^o, k\Delta y)}{k} \tag{7.9}$$

with an appropriate choice of d, the term

$$\frac{d \cdot g_{max} - (1 - k)g^{i}(y^{o})}{k} \geq 0 \quad \text{for } 0 < k < 1 .$$

That is, choose $d = 0$ when $g_{max} = 0$, inasmuch as $g_{max} = 0$ implies $g^{i}(y^{o}) \leq 0$, and choose $d = (1 - k)$ when $g_{max} > 0$, and recall $g_{max} \geq g^{i}(y^{o})$.

Now with our choice of $k = \delta/B_2$ and invoking (7.3), the condition (7.9) is satisfied at every step. But then when $d = (1 - k) = (1 - \delta/B_2)$, the infeasibility g_{max} must be reduced by at least $\delta/B_2 \cdot g_{max}$ at each step. Given $\varepsilon_1 > 0$ in at most a finite number of steps, $g_{max} < \varepsilon_1$ and remains so in all subsequent steps.

Further, at the conclusion of the solution of the local linear programming problems, we must have

$$\sum_{p} (-x_p)g^{p}(y^{o}) < -\varepsilon , \tag{7.10}$$

or we would satisfy condition 3 in our definition of a stationary point. Because the local linear programming problem is optimal, (4.4) and (4.5) yield,

$$\nabla g^{m}(y^{o})\Delta y = \sum_{p} (-x_p)g^{p}(y^{o}) + \bar{k} \sum_{p} (-x_p)r^{p}. \tag{7.11}$$

Employing these results, we can demonstrate a bounded reduction in the extremal function as follows:

$$g^{m}(y) = g^{m}(y^{o}) + \nabla g^{m}(y^{o})\Delta y \cdot k + R^{m}(k\Delta y)$$

[by 7.11]

$$= g^{m}(y^{o}) + \left[\sum_{p} (-x_p)g^{p}(y^{o}) + \bar{k} \sum_{p} (-x_p)r^{p}\right] k + R^{m}(k\Delta y)$$

[by (7.10), our choice of $\bar{k}$ and k, and (7.4)]

$$\leq g^m(y^o) + \left(-\varepsilon + \frac{\varepsilon}{B_3 + B_1 \cdot B_3} B_1 B_3\right)\frac{\delta}{B_2} + \frac{\varepsilon \cdot \delta}{B_2}\frac{1}{2(1 + B_1)}$$

[by algebraic simplification]

$$= g^m(y^o) - \frac{\varepsilon \cdot \delta}{B_2}\left[1 - \frac{B_1}{(1 + B_1)} - \frac{1}{2(1 + B_1)}\right]$$

$$= g^m(y^o) - \frac{\varepsilon \cdot \delta}{2B_2(1 + B_1)} \cdot \qquad (7.12)$$

Inasmuch as $g^m(y)$ is reduced by at least $\varepsilon\cdot\delta/[2B_2(1 + B_1)]$ at each step, we can contradict any bound on $g^m(y)$ in a finite number of steps. Hence we conclude that a stationary point is reached in at most a finite number of steps. QED.

8. INTERPRETATION OF THE STATIONARY POINT

To interpret a stationary point, we need an additional result. Consider any $y = y^o + \Delta y$ that is an ε-feasible solution of the constraints of (2.1), the original problem. By the Approximation Theorem

$$g^p(y) = g^p(y^o) + \nabla g^p(y^o)\Delta y + R^p(y^o, \Delta y) \leq \varepsilon_1 \quad p = 1, \ldots, m - 1 ,$$

or

$$\nabla g^p(y^o)\Delta y \leq -g^p(y^o) - R^p(y^o, \Delta y) + \varepsilon_1 . \qquad (8.1)$$

Now recall that for any solution $x_p \leq 0$ of the dual problem as stated in (4.5),

$$\sum_p x_p \nabla g^p(y^o) = \nabla g^m(y^o) . \qquad (8.2)$$

Using the results of (8.1) and (8.2), we can obtain a lower bound on the extremal function as follows:

$$
\begin{aligned}
g^m(y) &= g^m(y^o) + \nabla g^m(y^o)\Delta y + R^m(y^o, \Delta y) \\
&= g^m(y^o) + \left[\sum_p x_p \nabla g^p(y^o)\right]\Delta y + R^m(y^o, \Delta y) \\
&= g^m(y^o) + \sum_p x_p[\nabla g^p(y^o)\Delta y] + R^m(y^o, \Delta y) \\
&\geq g^m(y^o) + \sum_p (-x_p) g^p(y^o) + \sum_p (-x_p) R^p(y^o, \Delta y) \\
&\quad + R^m(y^o, \Delta y) + \sum_p x_p \varepsilon_1 \ . \qquad (8.3)
\end{aligned}
$$

Choose $\varepsilon_1 = \varepsilon/B_1$ and assume that condition (1) of a local linear stationary is not satisfied, then

$$\sum_p x_p \geq -B_1 \quad \text{and} \quad \sum_p x_p \varepsilon_1 \geq -\varepsilon \ .$$

Using this fact and recalling that global or local convexity for all functions would imply $R^i(y^o, \Delta y) \geq 0$. Formula (8.3) would then imply,

$$g^m(y) \geq g^m(y^o) + \sum_p (-x_p) g^p(y^o) - \varepsilon \ . \qquad (8.4)$$

When condition (3) in the definition of a stationary point obtains, we conclude

$$g^m(y^o) \leq g^m(y) + 2\varepsilon \quad \text{for all } y \ , \qquad (8.5)$$

and therefore by definitions (3.1) and (3.2) we have achieved a global or local ε-minimum. To demonstrate ε-inconsistency, the identical argument can be applied to a subproblem when condition (2) in the definition of a stationary point obtains. In fact, if the stronger condition

$$\sum_{p \in H} (-x_p) g^p(y^o) > -g^w(y^o) \qquad (8.6)$$

occurs at termination, the argument would demonstrate absolute inconsistency of the constraints. Without convexity it is theoretically possible to terminate at a "saddlepoint," although this is virtually impossible as a practical matter. The algorithm can be extended to a second-order method by explicit introduction of first-order necessary conditions as additional constraints [5]. It can also be extended to treat equality constraints.

9. AN ANALYTIC EXAMPLE PROBLEM

To illustrate some of these ideas, consider the following nonlinear programming problem.

Subject to

$$g^1(y) = y_1^2 + y_2^2 - 9 \leq 0$$

$$g^2(y) = -y_1 - y_2 + 1 \leq 0,$$

minimize

$$g^3(y) = (y_1 - 3)^2 + (y_2 - 2)^2 .$$

$$(y_i \geq 0)$$

Graphically, the problem becomes that shown in Fig. 3.

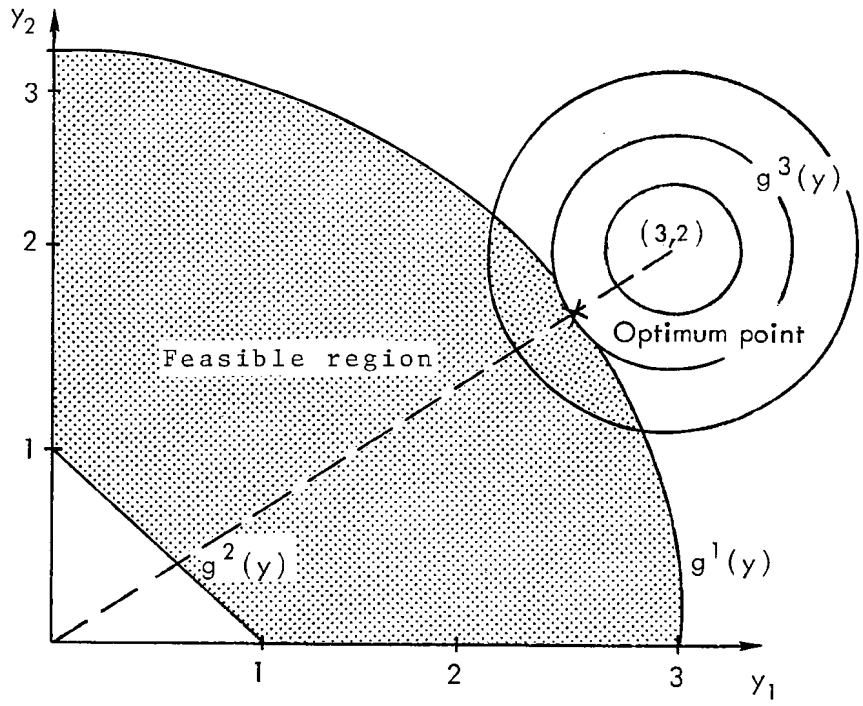


Fig.3 — Graphic solution of nonlinear example

The optimun occurs at the intersection of the line

$$2y_1 - 3y_2 = 0$$

connecting the centers of the circles $g^1(y)$ and $g^3(y)$ and the binding constraint circle $g^1(y) = 0$.

Eliminating y_1 from the above linear equation

$$y_1 = 3/2\ y_2$$

and substituting in $g^1(y) = 0$,

$$9/4\ y_2^2 = y_2^2 = 9$$

$$y_2 = 6/\sqrt{13}\ .$$

Plugging this value back into the expression for y_1,

$$y_1 = 3/2 \cdot 6/\sqrt{13} = 9/\sqrt{13} ;$$

or numerically

$$y_1^* = 2.49615$$

$$y_2^* = 1.6641 ,$$

and the optimum value is

$$g^3(y^*) = 0.366693 .$$

Now let us follow through the nonlinear programming algorithm. Since we must seek the solution on a bounded domain, we superimpose the range restrictions

$$0 \le y_1 \le 5 \quad \text{and} \quad 0 \le y_2 \le 5 ,$$

which are handled in the internal logic and not introduced explicitly (bounds of 3 would have been sufficiently high, but we want to avoid redundancy in this example). The associated local linear programming problems are the following :

Subject to

$$\nabla g^i(y^o)\Delta y \le -g^i(y^o) - \bar{k}r^i ,$$

minimize

$$\nabla g^m(y^o)\Delta y . \qquad \text{(See 2.3)}$$

In our problem,

$$\nabla g^1(y) = [2y_1, 2y_2]$$
$$\nabla g^2(y) = [-1, -1]$$
$$\nabla g^3(y) = [2(y_1 - 3), 2(y_2 - 2)] .$$

In general, r^i is taken as the error term $R^i(y^o, \Delta y)$ of the previous step and $\bar{k} = 1$. The error terms $R^i(y^o, \Delta y)$ reflect the length $|\Delta y|$ of the previous step and may cause the local linear programming problems to be inconsistent or prevent a local linear gain. As given in Lemma 1, however, the $\bar{k}$ can then be adjusted down to as low as

$$\bar{k} = \frac{\varepsilon}{(B_3 + B_1 B_3)}$$

parametrically in the course of the computation to gain consistency and a local linear gain. More specifically, if an inconsistency is encountered in the local linear programming problem, an index w will exist such that

$$[-g^w(y^o) - \bar{k}r^w] - \nabla g^w(y^o)\Delta y < 0 .$$

We employ (6.2) and calculate $\bar{k}$ to eliminate this violation, or

$$[-g^w(y^o) - \bar{k}r^w] - \left\{ \sum_{p \in H} x_p [-g^p(y^o) - \bar{k}r^p] \right\} = 0$$

$$\bar{k} = -\left[g^w(y^o) + \sum_{p \in H} (-x_p) g^p(y^o) \right] \Big/ \left[r^w + \sum_{p \in H} (-x_p) r^p \right] .$$

If

$$-\left[g^w(y^o) + \sum_{p \in H} (-x_p) g^p(y^o) \right] < \varepsilon ,$$

then rearranging the terms,

$$\sum_{p\epsilon H} (-x_p) g^p(y^o) > -g^w(y^o) - \varepsilon \ ,$$

and we terminate with the knowledge that the nonlinear problem is inconsistent inasmuch as we have satisfied condition (2) of a local linear stationary point. Note if

$$-\left[g^w(y^o) + \sum_{p\epsilon H} (-x_p) g^p(y^o)\right] \geq \varepsilon \ ,$$

then by our definition of the bounds B_1, B_2, and B_3,

$$\bar{k} \geq \frac{\varepsilon}{(B_3 + B_1 B_3)} \ ,$$

and we guarantee a movement of sufficient magnitude to achieve convergence. To insure a local linear gain, we must have

$$\nabla g^m(y^o) \Delta y < 0 \ .$$

Using the equivalent expression given in (4.5) yields

$$\sum_p (-x_p) g^p(y^o) + \bar{k} \sum_p (-x_p) r^p < 0 \ ,$$

or

$$\bar{k} < \frac{-\sum_p (-x_p) g^p(y^o)}{\sum_p (-x_p) r^p} \ .$$

The dual variables x_p required for these parametric adjustments are readily available with our L.P. code. This parametric adjustment will be illustrated in the course of solving the present problem.

Suppose we start at the point $y_o = (0, 0)$, and since we have no previous step estimates, take $r^1 = r^2 = 0$. The initial linear programming problem is the following:

Subject to

$$0\Delta y_1 + 0\Delta y_2 \leq 9$$
$$-\Delta y_1 - \Delta y_2 \leq -1 ,$$

minimize

$$-6\Delta y_1 - 4\Delta y_2 .$$

The implicit range restrictions on the variables are

$$0 \leq \Delta y_1 \leq 5 \quad \text{and} \quad 0 \leq \Delta y_2 \leq 5 .$$

The optimal solution (Fig. 4) is $\Delta y = (5, 5)$.

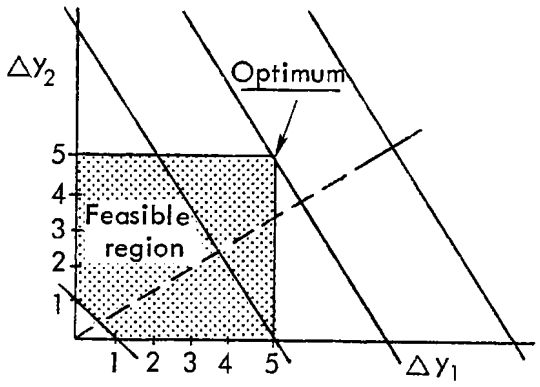


Fig.4 — Graphic solution of first L.L.P.

The new point will be of the form

$$y = y^o + k\Delta y = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + k \begin{bmatrix} 5 \\ 5 \end{bmatrix},$$

where we must determine $k \leq 1$ to reduce $g_{max} = 1$ or achieve feasibility. The error term $R^1(y^o, \Delta y)$ is obtained from

$$g^1(y) = g^1(y^o) + \nabla g^1(y^o)\Delta y + R^1(y^o, \Delta y)$$

by a single additional function evaluation of $g^1(y)$, inasmuch as $g^1(y^o)$ is already available, and $\nabla g^1(y^o)$ is given by the local linear programming solution. Carrying this out,

$$\begin{aligned} R^1(y^o, \Delta y) &= g^1(y) - g^1(y^o) - \nabla g^1(y^o)\Delta y \\ &= 41 - (-9) - 0 \\ &= 50 \ . \end{aligned}$$

Using the quadratic approximation,

$$g^1(y) = g^1(y^o) + [\nabla g^1(y^o)\Delta y]k + R(y^o, \Delta y)k^2$$

(which happens to be exact in this case) to solve for the k yields

$$-9 + 0 \cdot k + 50 \cdot k^2 = 0$$

$$\begin{aligned} k &= \frac{3}{\sqrt{50}} \\ &= 0.424264 \ . \end{aligned}$$

Now $R^2(y^o, \Delta y) = 0$ since the second constraint is linear and

$$\begin{aligned} g^2(y) &= g^2(y^o) + \nabla g^2(y^o)\Delta y \cdot k \\ &= 1 - 10 \cdot k \\ &\leq 0 \qquad \text{for } k \geq 0.1 \ . \end{aligned}$$

Hence, we are feasible for any value of k in the interval $0.1 \leq k \leq 0.424264$. We take k = 0.424264 which yields the best gain in $g^3(y)$.

The new point becomes

$$y^o = (2.121321, 2.121321)$$

and in setting up the next linear programming problem we have,

$$\begin{aligned} g^1(y^o) &= 0 \\ g^2(y^o) &= 2.242641 \\ R^1(y^o, \Delta y^o) &= 50 \\ R^2(y^o, \Delta y^o) &= 0 \ . \end{aligned}$$

The second linear program becomes the following:

Subject to

$$\begin{aligned} 4.24264\ \Delta y_1 + 4.24264\ \Delta y_2 &\leq -50 \\ -1.0\ \Delta y_1 - 1.0\ \Delta y_2 &\leq 3.24264\ , \end{aligned}$$

minimize

$$-1.75736\ \Delta y_1 + 0.242641\ \Delta y_2\ .$$

The range restrictions are

$$0 \leq y^o + \Delta y \leq 5$$

or

$$\begin{aligned} -2.121321 &\leq \Delta y_1 \leq 2.878679 \\ -2.121321 &\leq \Delta y_2 \leq 2.878679\ . \end{aligned}$$

Now, if we take both Δy_1 and Δy_2 at their lower bounds, the first linearized constraint becomes

$$4.24264 \cdot (-2.121321) + 4.24264 \cdot (-2.121321) = -18 > -50 ,$$

and the problem is clearly inconsistent. In the course of the solution of the linear programming problem, the initial $\bar{k} = 1$ is parametrically adjusted down to $\bar{k} = 0.275148$.

The right-hand side of the adjusted linear programming problem is

$$-g^1(y^o) - \bar{k}R^1(y^o, \Delta y^o) = -0 - \bar{k} \cdot 50 = -13.7574$$
$$-g^2(y^o) - \bar{k}R^2(y^o, \Delta y^o) = -g^2(y^o) - k \cdot 0 = 3.24264 .$$

The two linearized constraints were adjusted until they coincided to give a feasible region (Fig. 5). The unrefined optimal solution is

$$\Delta y_1 = -1.12133$$
$$\Delta y_2 = -2.121321 .$$

At this point the computer program makes a post-optimal adjustment of $\bar{k}$ to further improve the solution of the local linear program. The optimal solution of a linear program always occurs at a basic solution. The basis consists of n linearly independent row vectors or constraints where for theoretical purposes we include the range constraints of the form

$$\underline{e}_j \Delta y \leq U_j - y_j^o$$
$$\underline{e}_j \Delta y \leq y_j^o - L_j .$$

At the present optimal solution the basis is

$$B = \begin{bmatrix} 4.24264 & 4.24264 \\ 0 & -1 \end{bmatrix} .$$

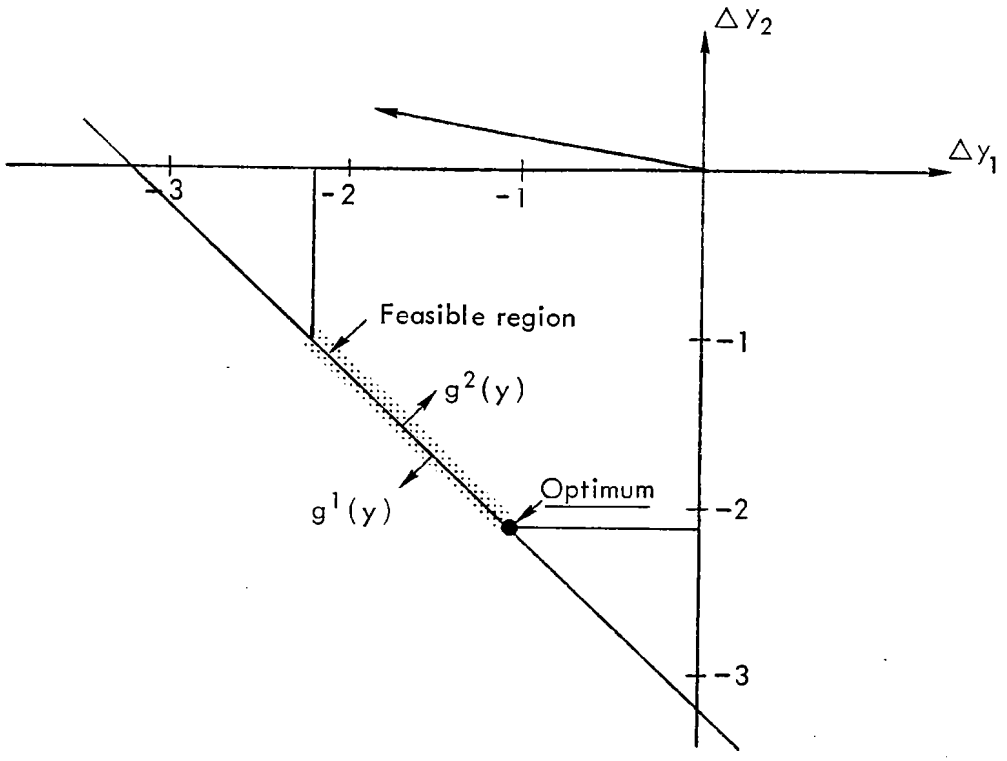


Fig.5 — Graphic solution of second L.L.P.

This consists of the first constraint and the lower-bound constraint on Δy_2.

The optimal solution is

$$\Delta y = B^{-1}(-g - \bar{k}r) \ .$$

In this particular instance,

$$B^{-1} = \begin{bmatrix} 0.235702 & 1 \\ 0 & -1 \end{bmatrix},$$

and

$$\Delta y = \begin{bmatrix} 0.235702 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2.121321 \end{bmatrix} + \bar{k} \begin{bmatrix} -50 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 2.121321 \\ -2.121321 \end{bmatrix} + \bar{k} \begin{bmatrix} -11.7851 \\ 0 \end{bmatrix} .$$

With our current adjusted $\bar{k} = 0.275148$ we obtain

$$\Delta y = \begin{bmatrix} -1.12133 \\ -2.121321 \end{bmatrix} .$$

Now observe,

$$\nabla g^3(y^o)\Delta y = (-1.75736,\ 0.242641 \left\{ \begin{bmatrix} 2.121321 \\ -2.121321 \end{bmatrix} + \bar{k} \begin{bmatrix} -11.7851 \\ 0 \end{bmatrix} \right\}$$

$$= -4.24264 + \bar{k} \cdot 20.71066 .$$

With our current adjusted $\bar{k} = 0.275148$,

$$\nabla g^3(y^o)\Delta y = 1.455856 ,$$

and since this is positive, we cannot even obtain or achieve a local gain. In general, as $\bar{k} \to 0$, we obtain the best "rate of gain," but the distance we can move approaches zero when we are on a boundary.

Suppose one of the constraining functions $g^i(y^o) = 0$ (on boundary) and we return to this boundary after movement,

$$g^i(y) = \nabla g^i(y^o)\Delta y \cdot k + R^i(y^o,\ k\Delta y) = 0 . \qquad (9.1)$$

From the local linear program the linear gain is

$$\nabla g^i(y^o)\Delta y = -\bar{k}r^i . \qquad (9.2)$$

Substituting expression (9.2) into expression (9.1) and using the approximation $R^i(y, k\Delta y) = k^2 \cdot R^i(y, \Delta y)$ yields

$$k[-\bar{k} \cdot r^i + k \cdot R^i(y^o, \Delta y)] = 0 ,$$

and we are at the boundary for k = 0 (no movement) and for

$$k = \bar{k} \frac{r^i}{R^i(y^o, \Delta y)} .$$

Now we neglect changes in the error term due to direction and position (with circles there is no error because they have uniform curvature). Since we always take $r^i = R^i(y^o, \Delta y^o)$, this assumption translates to

$$\frac{r^i}{R^i(y, \Delta y)} = \frac{R^i(y^o, \Delta y^o)}{R^i(y, \Delta y)} = \frac{\Delta y^{oT} \Delta y^o}{\Delta y^T \Delta y} ,$$

and the distance moved becomes

$$k = \frac{\bar{k}(\Delta y^{oT} \Delta y^o)}{(\Delta y^T \Delta y)} .$$

Assuming we move this distance, the objective is to minimize

$$g^m(y) = g^m(y^o) + \nabla g^m(y^o)\Delta y \cdot k + R^m(y^o, \Delta y)k^2 .$$

And employing our assumption that

$$R^i(y^o, \Delta y) = \frac{\Delta y^T \Delta y}{\Delta y^{oT} \Delta y^o} R^i(y^o, \Delta y^o) ,$$

as well as our expression for k yields

$$g^m(y) = g^m(y^o) + \nabla g^m(y^o)\Delta y \frac{\bar{k}(\Delta y^{o^T}\Delta y^o)}{\Delta y^T \Delta y}$$

$$+ \frac{\bar{k}^2(\Delta y^{o^T}\Delta y^o)^2}{(\Delta y^T \Delta y)^2} \frac{\Delta y^T \Delta y}{\Delta y^{o^T}\Delta y^o} R^m(y^o, \Delta y^o)$$

$$= g^m(y^o) + \frac{[\nabla g^m(y^o)\Delta y \cdot \bar{k} + R^m(y^o, \Delta y^o)\bar{k}^2]\Delta y^{o^T}\Delta y^o}{\Delta y^T \Delta y} .$$

Now defining

$$\underline{d} = B^{-1}g \quad \text{and} \quad \underline{p} = B^{-1}r^i ,$$

we have

$$\Delta y = \underline{d} + \bar{k}\underline{p}$$

and

$$\Delta y^T \Delta y = \underline{d}^T\underline{d} + 2\underline{d}^T\underline{p} \cdot \bar{k} + (\underline{p}^T \cdot p)\bar{k}^2 .$$

Further, defining $\tilde{d}_m = \nabla g^m(y^o)\underline{d}$, $\tilde{p}_m = \nabla g^m(y^o)\underline{p}$

$$\nabla g^m(y^o)\Delta y = \nabla g^m(y^o)\underline{d} + \bar{k}\nabla g^m(y^o)\underline{p} = \tilde{d}_m + \bar{k}\tilde{p}_m .$$

With the additional definition of

$$\tilde{\tilde{p}}_m = \tilde{p}_m + R^m(\Delta y^o) ,$$

we can rewrite the expression for $g^m(y)$ as

$$g^m(y) = g^m(y^o) + \frac{(\tilde{d}_m\bar{k} + \tilde{\tilde{p}}_m\bar{k}^2)(\Delta y^{o^T}\Delta y^o)}{\underline{d}^T d + 2\underline{d}^T p\bar{k} + \underline{p}^T p\bar{k}^2} .$$

To obtain the minimum value of $g^m(y)$, we set the derivative

$$\frac{d\, g^m(y^o)}{d\, \bar{k}} = 0$$

and solve for $\bar{k}$.

Working through the algebra

$$\bar{k}^* = \frac{-B + \sqrt{B^2 - AC}}{A},$$

where

$$A = 2(\underline{d}^T\underline{p}) \cdot \tilde{\tilde{p}}_m - \tilde{d}_m(\underline{p}^T\underline{p})$$
$$B = (\underline{d}^T\underline{d})\tilde{\tilde{p}}_m$$
$$C = (\underline{d}^T\underline{d})\tilde{d}_m .$$

In our example,

$$(\underline{p}^T\underline{p}) = 138.88882$$
$$(\underline{d}^T\underline{p}) = -25$$
$$(\underline{d}^T\underline{d}) = 9$$
$$\tilde{\tilde{p}}_m = 70.710663$$
$$\tilde{d}_m = -4.24263$$
$$A = -2946.286$$
$$B = 636.39746$$
$$C = -38.183762$$
$$\bar{k}^* = 0.0324352 .$$

Using the optimal $\bar{k}^*$ yields

$$\Delta y = \begin{bmatrix} 2.121321 \\ -2.121321 \end{bmatrix} + 0.0324352 \begin{bmatrix} -11.7851 \\ 0 \end{bmatrix} = \begin{bmatrix} 1.739072 \\ -2.121321 \end{bmatrix},$$

and the adjusted right-hand side of the underlying linear program (Fig. 6) becomes

$$-g^1(y^o) - \bar{k}^* R^1(y^o, \Delta y^o) = 0 - \bar{k} \cdot 50 = -1.62175$$
$$-g^2(y^o) - \bar{k}^* R^2(y^o, \Delta y^o) = -g^2(y^o) - \bar{k} \cdot 0 = 3.24264 .$$

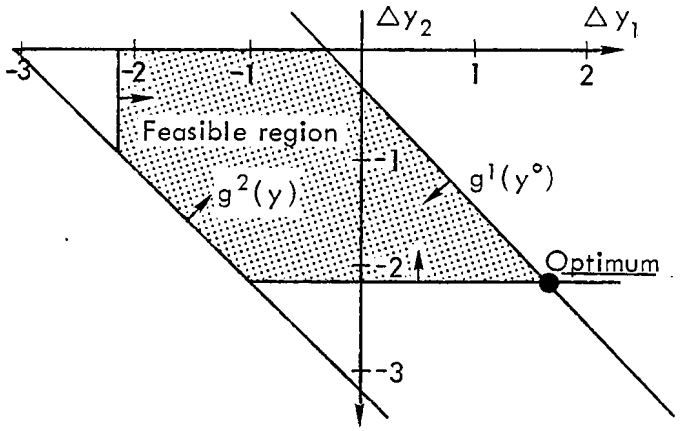


Fig.6 — Graphic solution of second L.L.P. after parametric adjustment

Again, using the formula

$$R^1(y, \Delta y) = g^1(y) - g(y^o) - \nabla g^1(y^o)\Delta y ,$$

we calculate

$$R^1(y, \Delta y) = 7.524391 .$$

Using the quadratic approximation

$$g^1(y) = g^1(y^o) + [\nabla g^1(y^o)\Delta y]k + R(y, \Delta y)k^2$$

(which is exact in this case) to solve for k yields

$$0 + 1.621761 \cdot k + 7.524391\, k^2 = 0$$

$$k = \frac{1.621761}{7.524391}$$

$$= 0.2155338 .$$

Now $R^2(\Delta y) = 0$ because the second constraint is linear and

$$g^2(y) = g^2(y^o) + \nabla g^2(y^o)\Delta y \cdot k$$

$$= -3.242641 + 0.3822479 \cdot k$$

$$= 0$$

for $k \leq 8.483084$. Hence, we are feasible for $0 \leq k \leq 0.2155338$, and the optimum value of $g^3(y)$ on this interval occurs for $k = 0.2155338$.

Using this value of k, we calculate

$$y = y_o + k\Delta y = \begin{bmatrix} 2.121321 \\ 2.121321 \end{bmatrix} + 0.215534 \begin{bmatrix} 1.739072 \\ -2.121321 \end{bmatrix}$$

$$= \begin{bmatrix} 2.496150 \\ 1.664104 \end{bmatrix},$$

the optimum solution. The optimum value is

$$g^3(y) = 0.3666912 .$$

It should be observed that in making the parametric adjustments to $\bar{k}$, we may transfer basis in the linear programming tableau. This requires calculating the range of adjustment of $\bar{k}$ to preserve the current basis and making additional pivots if we adjust beyond these limits.

IV. PROGRAM INPUT, OUTPUT AND EXAMPLE

1. INPUT

Except for the last card, input for the program is in the free format "LIST" style. Decimal points need not be punched for numbers having integer values. One or more spaces separate entries on a data input card. As the program is currently written, an input record consists of one, and only one, punched card. Only columns 1 through 72 are processed; columns 73 through 80 are ignored.

Let the following symbols (which are the same in the program) be defined:

NAT = number of aircraft types (=I)
NTT = number of target types (=J)
NADC = number of side constraints.

The input for the program consists of the following cards:

1. One card with the title of the problem punched in columns 1 through 72.
2. One card with the values for $S_1, \ldots, S_{NAT}$ (sortie limits). The number of nonzero numbers punched on this card defines the value of NAT, whose maximum value is 10, and whose minimum is 1.
3. NTT cards, each giving the data for one target type, and the target type being given the number of the card in the order read. For card j of the set, the data assigned to target j are read from the card in the order

$$T_j, D_j, \ell_j, C_j, V_j, P_{1j}, \ldots, P_{NATj} .$$

Each card would normally have NAT + 5 numbers punched in it. If it has less than this, the program supplies zeroes for the missing data. T_j must be nonzero on every card. The maximum number of cards is 45, and the minimum is 1.

4. The next card is blank. This signals the end of the target data read in step 3 and permits the value of NTT to be defined. This card is equivalent to a card in step 3 being read with a value of $T_j = 0$.
5. NADC cards, each card giving the data for one side constraint. Each card has data giving "+" for an upper-bound constraint and "-" for a lower-bound constraint, followed immediately by the aircraft type to which it applies. There is no blank between the sign and the aircraft type. Then, after one or more blanks, the card contains the percentage of sorties (expressed as a decimal fraction) applicable to this constraint, and this is followed by a list of the target types (a maximum of 10) that are affected by this constraint. The maximum number of cards in this step is 50, and the minimum is zero.
6. The next card is blank. This allows NADC to be defined as the number of cards read in step 5, similar to the way NTT was defined in step 4.
7. The last card is the card with the parameters for the nonlinear program. This card generally does not change for a particular deck compilation, and its parameters depend on the parameter sizes and options in the underlying nonlinear program. With the current version of the program, this card should be

IX1 = 0 ;

which is the parameter that controls the amount of intermediate output.

Output corresponding to the input is produced for cross-checking purposes.

At this point the program will attempt to solve the problem presented and will display the output when computation is completed.

Only one problem may be solved in any particular computer run. Since this is the case, the card of step 7 should be the last card

in the input data set and should be followed by the end of data set delimiter card.

2. INPUT FOR A SAMPLE PROBLEM

The attached exhibit (Fig. 7) titled

Sample Data of
Sortie Allocation Model

will be used to illustrate the input and the output of the computer program. The input for the computer program consists of 27 of the 29 cards shown in Fig. 8. Two of the cards listed are not input cards, they are Job Control Language cards required by the IBM 370/158 operating system. In particular, the card on line 1 (//GO.SYSIN DD *) signals the beginning of the input data set, and the card on line 29 (/*) signals the end of the data set.

Line 2 is the title card; it may contain any title that the user wishes repeated on the output. Columns 1 through 72 are used for input and columns 73 through 80 are ignored in this, and all other cards to follow.

Line 3 is a list of the sortie limits; we have used $0.9S_i$ as given below in the Sample Data. Note that there are 4 entries: This defines the number of aircraft types (NAT) to be 4.

Lines 4 through 13 are the 10 cards needed giving the information on the 10 target types. The information is exactly in the order given as the Target Type data in the Sample Data, with the exception that the subscript j is not punched, and explicit zeroes replace the positions where no entry (--) is made.

Line 14 is the blank card that terminates the Target Type data.

Lines 15 through 20 are the 6 cards needed for the upper-bound sortie constraints. They are punched exactly as the information shown in the Sample Data.

Lines 21 through 26 are the 6 cards needed for the lower-bound sortie constraints. The first number in each line is preceded by a minus sign in order to distinguish the lower-bound from the upper-bound

Target Type	j	T_j	D_j	ℓ_j	C_j	V_j	P_{1j}	P_{2j}	P_{3j}	P_{4j}
Building	1	598	9	0	10^{-5}	0.97	0.516	0.521	0.153	--
Ground control interceptor radar	2	663	0	0	10^{-5}	2.08	0.990	1.137	1.087	--
Petroleum, oil, and lubrication refinery	3	24	0	0	10^{-5}	0.37	0.148	0.152	0.117	--
Air base shelter	4	2227	0	0	1	3.13	0.343	0.513	0.462	--
Aircraft revetted	5	1666	0	0	1	3.13	6.183	6.555	7.545	--
SAM site	6	190	0	0	10^{-5}	3.77	2.630	2.501	1.046	--
Tanks	7	1260	0	0	10^{-5}	1.00	0.816	0.804	--	1.997
Trucks	8	1358	0	0	10^{-5}	0.95	0.627	0.408	--	0.584
Mobile SAM site	9	18	0	10	10^{-5}	3.43	0.725	0.573	--	0.423
Personnel in foxhole	10	857	0	0	10^{-5}	0.51	0.262	0.205	--	0.377

Aircraft Type	i	S_i
A-7	1	200
F-4D	2	400
F-111	3	500
AX	4	200

Upper-Bound Sortie Constraints					Lower-Bound Sortie Constraints				
Aircraft Type	Max % Sortie	Target	Set	J	Aircraft Type	Max % Sortie	Target	Set	J
1	80	2	6	9	1	10	1	4	5
2	80	2	4		2	10	3	6	
3	80	2	5		3	10	1	4	
4	80	7			4	10	9	10	
1	80	1	4	5	1	10	8	10	
3	80	1	4		3	10	2	4	5

e.g., $S_{1,2} + S_{1,6} + S_{1,9} \leq 0.8\ S_1$ e.g., $S_{1,1} + S_{1,4} + S_{1,5} \geq 0.1\ S_1$

Fig. 7 — Sample data of Sortie Allocation Model

```
1.      //GO.SYSIN DD *
2.            WEAPONS MIX  ...  SMALL TEST PROBLEM #2
3.         100 360 450 180
4.        598. 9     0 1.E-5 0.97 0.516 0.521 0.153 0.
5.        663. 0     0 1.E-5 2.08 0.990 1.137 1.087 0.
6.         24. 0     0 1.E-5 0.37 0.148 0.152 0.117 0.
7.        2227 0     0 1. 3.13 0.343 0.513 0.462 0.
8.        1666 0     0 1. 3.13 6.183 6.555 7.545 0.
9.        190. 0     0 1.E-5 3.77 2.630 2.501 1.046 0.
10.       1260 0     0 1.E-5 1.00 0.816 0.804 0. 1.997
11.       1358 0     0 1.E-5 0.95 0.627 0.400 0. 0.584
12.        13.  0    10 1.E-5 3.43 0.725 0.573 0. 0.423
13.       857. 0     0 1.E-5 0.51 0.262 0.205 0. 0.377
14.
15.     1 .8  2 6 9
16.     2 .8  2 4
17.     3 .8  2 5
18.     4 .8  7
19.     1 .8  1 4 5
20.     3 .8  1 4
21.     -1 .1  1 4 5
22.     -2 .1  3 6
23.     -3 .1  1 4
24.     -4 .1  9 10
25.     -1 .1  8 10
26.     -3 .1  2 4 5
27.
28.      IX1=0;
29.     /*
```

Fig. 8 — Sample input data cards

constraints. Otherwise these data are exactly the same as the information shown in the Sample Data.

Line 27 is the blank card that ends the data for the side constraints.

The last data card, on line 28, sets the value of IX1 to zero. This is the only card that requires a semicolon to end the input of the card; the other cards (except the title card) do not contain punctuation (other than decimal points) and automatically end in column 72. The value of IX1 may be 0, 1, 2, 3, or 4, and this number controls the amount of intermediate output that will be generated. Since intermediate output is used primarily for debugging purposes, we have set the value to zero, which indicates that we wish no intermediate output.

3. OUTPUT

Much of the program output is self-evident from the description of the input and from the problem definition. The output will

usually occupy eight pages; for a small problem such as the sample problem above, much of each page will be blank. The output produced specifically for the type of problem we are concerned with (as opposed to a general nonlinear programming problem) is divided into eight logical pages: each logical page will occupy at least one physical page, perhaps more.

In general, the output the program produces is as follows:

Logical Page	Contents of Page
1	Target type data input and sortie limit input.
2	Input for side constraints; Type = "UP" for upper bound and "LO" for lower bound. Also prints problem parameters (see below).
3	Intermediate output of nonlinear programming problem; in the Sample Problem above, this page contains only a list of the nonlinear program's default parameters.
	The remainder of the output consists of the optimal primal and dual solution.
4	Solution: optimal value and multipliers for the Target Type constraints and the Aircraft Type constraints.
5	Solution: multipliers for the side constraints.
6	Solution: number and total sorties by aircraft and target types.
7	Solution number of kills calculated from the sorties.
8	Final output of the nonlinear programming routine. The multipliers and primal variables are listed by variable number. This output is redundant and may be used for cross-checking. The value for "TCK" should be a small negative number or zero to indicate numerical convergence. In general, convergence is indicated by $-TL < TCK \leq 0$, where TL is printed on page 1 and has the value 0.0001. In the Sample Problem, convergence was exact, so that "TCK" is zero.

The parameters on logical page 2 are NAT, NTT, NADC, MNAT, MNTT, and MNADC. Of these, NAT, NTT, and NADC are as defined in the input, while MNAT, MNTT, and MNADC are the maximum values that the dimensions of the program will allow for NAT, NTT, and NADC, respectively.

4. OUTPUT FOR A SAMPLE PROBLEM

The 8-page output from the sample problem given above is shown in Fig. 9. The format is exactly as above for the general case.

WEAPONS MIX ... SMALL TEST PROBLEM #2 PAGE 1

TARGET	T	C	LB	CC	VALUE	PROB...			
1	598	9	0	0.0000100	0.970	0.5160	0.5210	0.1530	0.0000
2	663	0	0	0.0000100	2.080	0.9900	1.1370	1.0870	0.0000
3	24	0	0	0.0000100	0.370	0.1480	0.1520	0.1170	0.0000
4	2227	0	0	1.0000000	3.130	0.3430	0.5130	0.4620	0.0000
5	1666	0	0	1.0000000	3.130	6.1830	6.5550	7.3450	0.0000
6	190	0	0	0.0000100	3.770	2.0300	2.5010	1.0460	0.0000
7	1260	0	0	0.0000100	1.000	0.8160	0.8040	0.0000	1.9970
8	1356	0	0	0.0000100	0.950	0.6270	0.4080	0.0000	0.5840
9	18	0	10	0.0000100	3.430	0.7250	0.5730	0.0000	0.4230
10	857	0	0	0.0000100	0.510	0.2020	0.2050	0.0000	0.3770

SORTIE LIMITS
180.000000 360.000000 450.000000 180.000000

WEAPONS MIX ... SMALL TEST PROBLEM #2 PAGE 2

SIDE CONSTRAINTS

#	TYPE	A/C	PCT	TARGETS...
1	UP	1	0.80000	2 6 9
2	UP	2	0.80000	2 4
3	UP	3	0.80000	2 5
4	UP	4	0.80000	7
5	UP	1	0.80000	1 4 5
6	UP	3	0.80000	1 4
7	LO	1	0.10000	1 4 5
8	LO	2	0.10000	3 6
9	LO	3	0.10000	1 4
10	LO	4	0.10000	5 10
11	LO	1	0.10000	8 10
12	LO	3	0.10000	2 4 5

CONSTRAINT COUNT	MAXIMUM	NUMBER
AIRCRAFT TYPES	10	4
TARGET TYPES	45	10
SIDE CONSTRAINTS	50	12

WEAPONS MIX ... SMALL TEST PROBLEM #2 PAGE 3

M= 37 M2= 0 N3= 40 N2= 0 ICYCLE= 25
IX1= 0 IX2= 0 KERNF= 100 JM= 0
ZL= 9.999999999999999E-09
RELAX=-1.000000000000000E+00 IRD= 0 IWR= 0 NEGF= 1
EB= 0.000000000000000E+00 TL= 9.999999999999999E-05
AR= 1.000000000000000E+03 OG= 1.000000000000000E+00
NP= 4;

Fig. 9 — Sample output

WEAPONS MIX ... SMALL TEST PROBLEM #2 PAGE 4

ITERATION 10 OBJECTIVE VALUE 6533.46401

MARGINAL VALUES ON KILL LIMITS

TARGET	UPPER BOUND	LOWER BOUND
1	0.000000	0.000000
2	0.000000	0.000000
3	0.000000	0.000000
4	0.000000	0.000000
5	0.000000	0.000000
6	2.969157	0.000000
7	0.000000	0.000000
8	0.000000	0.000000
9	0.524976	0.000000
10	0.000000	0.000000

MARGINAL VALUES ON AIRCRAFT TYPE AVAILABILITY

A/C TYPE	VALUE
1	2.106116
2	2.232831
3	1.419311
4	1.228810

WEAPONS MIX ... SMALL TEST PROBLEM #2 PAGE 5

MARGINAL VALUES ON SIDE CONSTRAINTS

#	TYPE	A/C	VALUE
1	UP	1	0.000000
2	UP	2	0.132117
3	UP	3	1.150743
4	UP	4	0.768185
5	UP	1	0.000000
6	UP	3	0.000000
7	LO	1	0.000000
8	LO	2	0.230018
9	LO	3	0.000000
10	LO	4	0.000000
11	LO	1	1.510467
12	LO	3	0.000000

WEAPONS MIX ... SMALL TEST PROBLEM #2 PAGE 6

SORTIES BY A/C TYPE

TARGET TYPE	TOTAL	1	2	3	4
1	0.000	0.000	0.000	0.000	0.000
2	288.000	0.000	288.000	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000
4	90.000	0.000	0.000	90.000	0.000
5	516.167	120.167	36.000	360.000	0.000
6	74.009	38.009	36.000	0.000	0.000
7	144.000	0.000	0.000	0.000	144.000
8	18.000	18.000	0.000	0.000	0.000
9	39.824	3.824	0.000	0.000	36.000
10	0.000	0.000	0.000	0.000	0.000
TOTAL	1170.000	180.000	360.000	450.000	180.000

Fig. 9 — continued

WEAPONS MIX ... SMALL TEST PROBLEM #2 PAGE 7

TARGET TYPE	TOTAL	1	2	3	4
1	9.000	0.000	0.000	0.000	0.000
2	327.455	0.000	327.455	0.000	0.000
3	0.000	0.000	0.000	0.000	0.000
4	41.194	0.000	0.000	41.194	0.000
5	1484.693	298.525	94.815	1091.349	0.000
6	190.000	99.964	90.036	0.000	0.000
7	287.568	0.000	0.000	0.000	287.568
8	11.286	11.286	0.000	0.000	0.000
9	18.000	2.772	0.000	0.000	15.228
10	0.000	0.000	0.000	0.000	0.000

WEAPONS MIX ... SMALL TEST PROBLEM #2 PAGE 8

```
LAGRANGE MULTIPLIERS
11,    2.1061172CE+0C 12,    2.23283168E+00 25,    1.51046725E+00
 6,    2.96915728E+0C 17,    1.15C74438E+00 18,    7.6818d992E-01
14,    1.22881045E+0C 13,    1.41931130E+00  9,    5.24976456E-01
16,    1.32116643E-01 22,    2.30C18323E-01
ITC1=  10  IPC=   1  IPN=  40
SLPG=   0.000000CE+00  TCK=   0.0000000E+00  G(M)=  -6.5334640E+03
PRIMAL VARIABLES
  5            1.2016694E+02
  6            3.8009499F+01
  8            1.7999994E+01
  9            3.8235674E+C0
 12            2.8799998E+02
 15            3.60CCC3CE+01
 16            3.5999987E+01
 24            9.0000021F+01
 25            3.5999998E+02
 37            1.4399999F+02
 39            3.6000CC9E+01
```

Fig. 9 — continued

Appendix A
DETAILS OF THE NONLINEAR PROGRAM

The nonlinear program solves the following problems: subject to the constraints

$$g^i(y_1, y_2, \ldots, y_{N3}) = 0 \quad \text{for } i = 1, 2, \ldots, M2$$

and

$$g^i(y_1, y_2, \ldots, y_{N3}) \leq 0 \quad \text{for } i = M2 + 1, M2 + 2, \ldots, M - 1 ,$$

minimize the function

$$g^M(y_1, y_2, \ldots, y_{N3})$$

with

$$y_1, y_2, \ldots, y_{N2} \text{ unrestricted}$$

and

$$y_{N2+1}, y_{N2+2}, \ldots, y_{N3} \text{ nonnegative.}$$

The above variables are also subject to default bounds; the defaults are lower bound of -100 and upper bound of +100. These bounds may be set in a user procedure (SETUP or RESET) to any numbers consistent with the type of problem being solved.

The constants M2, M, N2, N3 appear in the program as in the above equations. It is permissible for M2 and N2 to be zero; this results in an absence of equality constraints or free variables, respectively. Also there may be no constraints at all, in which case M = 1. The independent variables $y_1, y_2, \ldots, y_{N3}$ appear in the program as the vector Y(·). The only explicit restriction on the g^i functions is

that their derivatives should exist and be continuous. An evaluation of the g^i functions is stored in the program in the vector G(·) after being evaluated in the procedure FCNGEN. The program also evaluates the gradient of the g functions. It does this one variable at a time--to obtain the partials of the g functions with respect to y_j, the program calls the procedure COLGEN(j). The resulting derivatives are stored in the CA(·) vector.

Communication among the various procedures is done via global variables (or EXTERNAL variables in PL/1 programs). Brief descriptions of these variables are given in Tables 1 and 2. Most have a distinct operational use; however, some may be used for temporary storage or for special applications. Certain nondimensioned variables may be input via a "GET DATA" statement; these need only be input if the user wants to override the defaults of the program. (An exception to this is that at least one parameter must be read in; this application uses the parameter IX1, which controls the amount of intermediate printout.) Some of the parameters in the "GET DATA" statement are global variables, while others are used only in the main routine and are nonglobal. Not every one of these parameters is applicable to the general continuous nonlinear programming problem described here.

The remainder of this section contains (1) dimensioned global variables, (2) nondimensioned global variables, (3) input parameters, (4) nonlinear subroutine calls, (5) flow of main procedure, (6) functions of the individual procedures, (7) logic for procedure INVDET, and (8) logic of column selection when KO(3) is being determined.

DIMENSIONED GLOBAL VARIABLES

Variable	Meaning
Y(.)	Values of the primal variables.
CA(.)	One column of the gradient as produced by procedure COLGEN.
G(.)	The evaluated function values as produced by procedure FCNGEN.
V(.,.)	The kernel of the inverse of the local linear programming problem.
UB(.)	The upper-bound vector associated with the variables.
BL(.)	The lower-bound vector for variables.
FL(.)	The linear correction term $\nabla g^i(y^o)\Delta y^o$.
R(.)	The estimate of the remainder $R^i(y^o, \Delta y)$.
PT(.)	Temporary P, used in procedure POAD.
DEL(.)	The computed change in the primal variables.
G1(.)	Storage for the previous value of G.
LGV(.)	Logical vector, denotes accessibility of variables.
IPR(.)	Permutation vector, allows order of variables to be changed.
I(.)	Permutation vector of row numbers.
J(.)	Permutation vector for basic columns.
K(.)	Control sequence (used in blocking).
LGB(.)	Logical bound: variable at lower bound = -1, at upper bound = +1, between = 0.
IS(.)	Scanning sequence in procedure LINPRG.

NONDIMENSIONED GLOBAL VARIABLES

Variable	Meaning
ZL*	Zero level for variables.
TL*	Termination level, "ε."
VALID	Termination check calculation in main routine.
SUPG	Value of the maximum infeasibility.
BEND	Bend factor, $\bar{k}$ in the theoretical development.
SC	Sliding factor used in approximation range calculation.
M*	Number of constraints plus one (for objective).
N3*	Number of variables.
M2*	Number of equality constraints (default 0).
N2*	Number of free variables (default 0).
ISF	Stop flag. Set to 1 when terminating.
KFFQ*	Kernel flush frequency. The parameter KERNF is zero except on iteration multiples of this parameter at which time KERNF = 1000. If KFFQ = 0, KERNF is always zero. Default 0.
IX1*	Amount of intermediate output; 0, 1, 2, 3, 4. Default 1.
IX2*	Default to 0 for general problem. If 1, solves linear problem.
IPC*	First permutation level to calculate. Default 1.
IPN*	Last permutation level to calculate.
N	Number of columns in explicit tableau.
LH	Number of rows in kernel.
ICYCLE*	Iteration limit. Default 25.
KERNF	Lower bound on kernel size before flush.
LT	Index of row for pivot.
LS	Index of column for pivot.
ITC1	Outside iteration counter.
ITC	Number of pivots in local linear program.
NEGF*	Cost selection rule: most negative = 1, default; first negative = 0.
M1	Number of binding equations in basis.
M3	Number of constraints (M - 1).

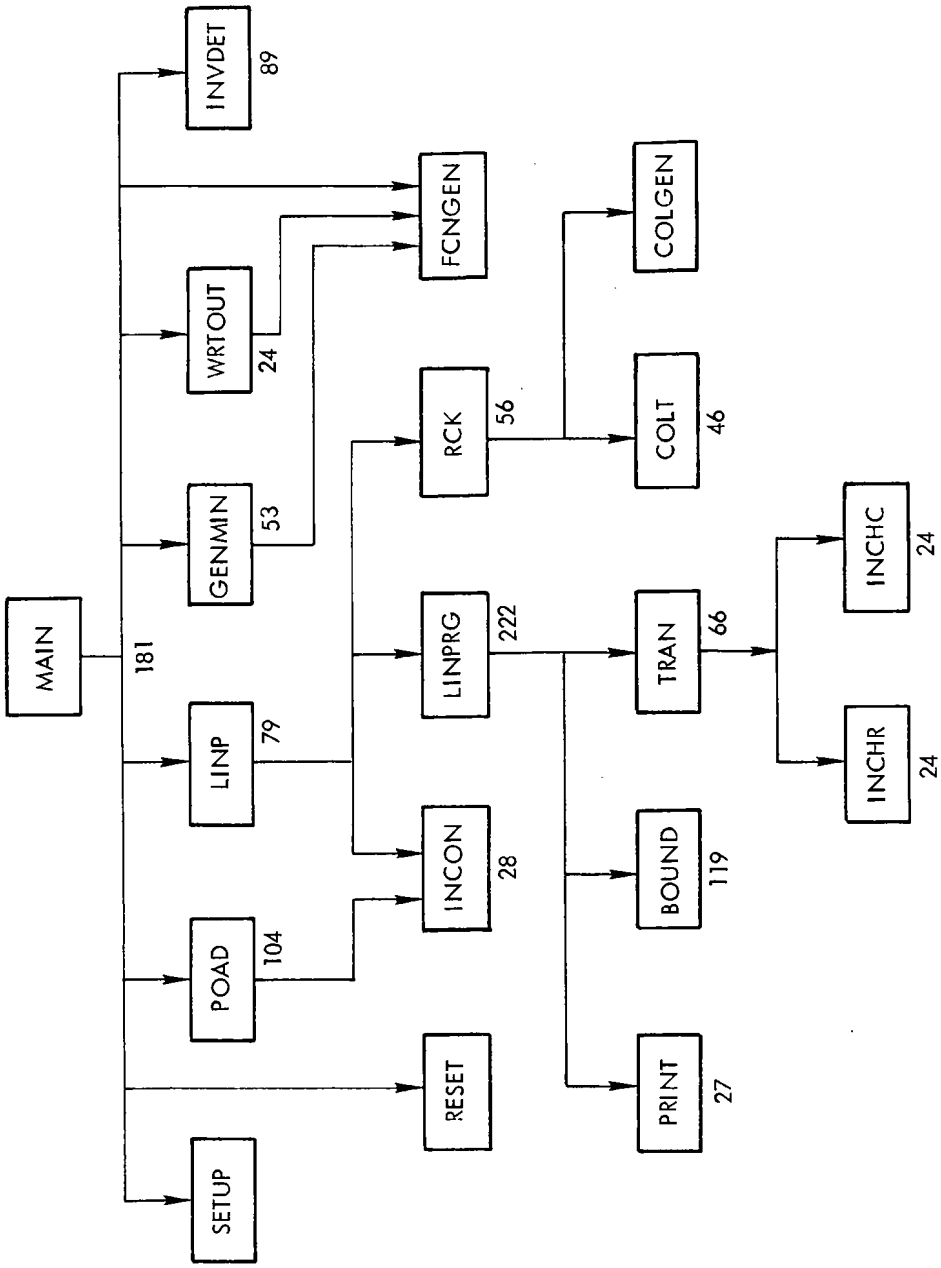
* denotes input parameter.

NONDIMENSIONED GLOBAL VARIABLES (continued)

Variable	Meaning
N1	Number of free variables in dual basis (driven from B to K).
NO	Length of the control sequence, K.
NS	Length of the scanning sequence, IS.
NMAX	Maximum size of LH, currently 114.

NONGLOBAL INPUT PARAMETERS

Parameter	Meaning
JM	JM=1 means that a search for minimum will be done among NP points; JM=0 means that a quadratic approximation is used to find the minimum. In this application, JM=0 is the default.
RELAX	(Default = -1.) Used in the termination check calculation.
EB	Equation Bandwidth. Used with nonlinear equality constraints to leave space between the two inequalities. Default to zero.
AR	Approximation Range. Used when the steps are large and wild. Default is 10.
NP	Number of points used with JM=1 option. Default is 4.



Numbers indicate size of procedure in number of PL/1 statements

Fig. 10 — Nonlinear subroutine calls

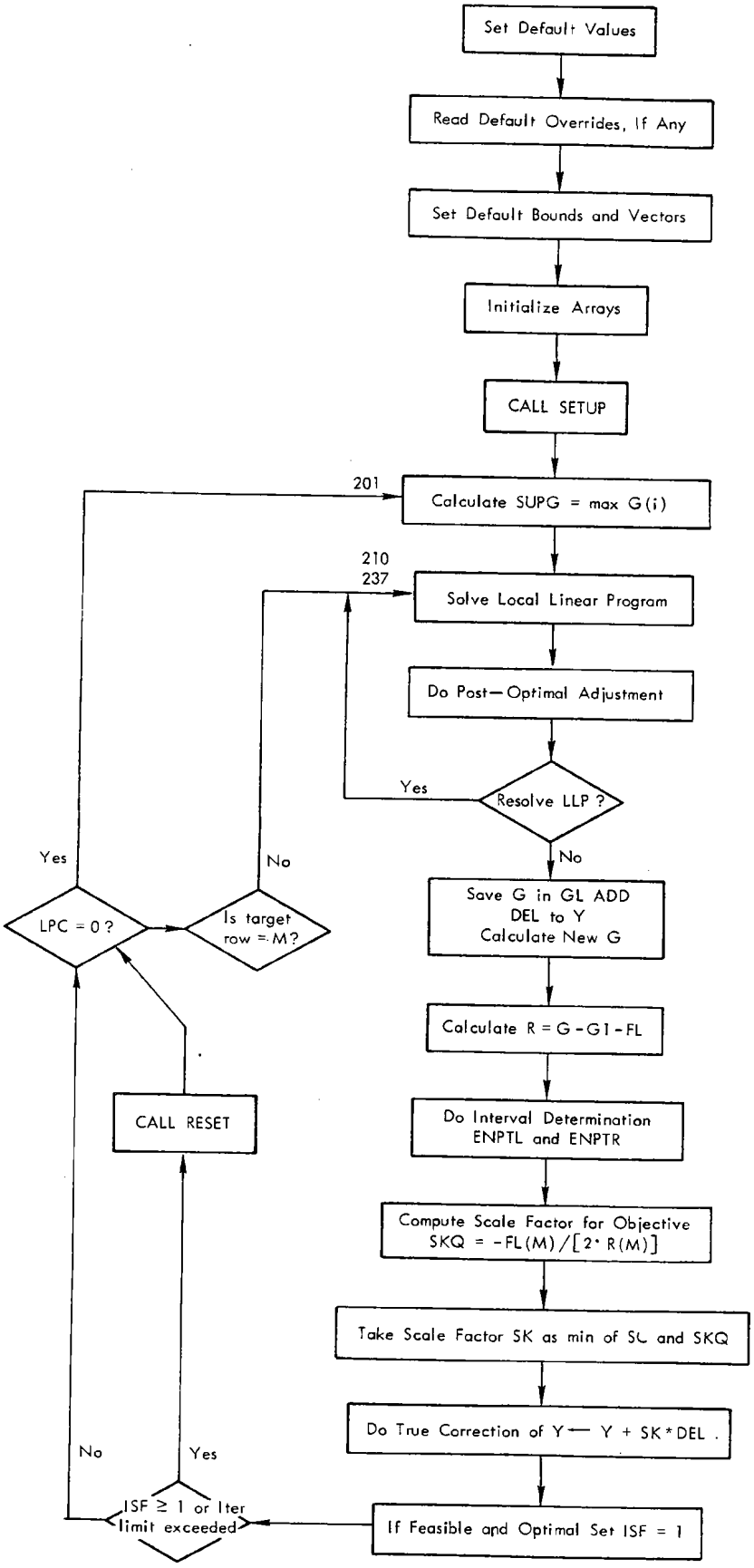


Fig. 11 — Main procedure

FUNCTIONS OF THE INDIVIDUAL PROCEDURES

Procedure	Meaning
MAIN	Central control for the flow of logic needed to solve the general problem.
POAD	(Post-Optimal Adjustment.) Adjusts the parameters for the nonlinear step. Does the bookkeeping for the variables at upper and lower bounds, using vector LGB. Calculates Δy (DEL) and then finds $\bar{k}$ (BEND).
INVDET	(Interval Determination.) Uses a quadratic approximation on each constraint to determine feasible interval of movement. If the intersection of all the intervals is null, the main program decreases the required infeasibility by a factor of two (up to 32 times) and then redoes the calculation for a feasible interval.
GENMIN	Does a search on the feasible interval for a minimum objective function value (option for when JM > 0).
LINP	Main control for solving the local linear programming problem.
RCK	(Row Check.) Checks the target row [K(2)] for the first negative element (if NEGF = 0) for the maximum negative element (if NEGF $\neq$ 0). If one is found, the column is designated JH.
COLT	(Column Transformation.) Transforms column in the CA vector and places it in the N-1st column of V.
INCON	Calculates (if IEX $\leq$ 0) the transformed R vector and places it in PT. If IEX > 0, adjusts the transformed right-hand side by TM*PT(·). If IEX = 0, calculates PT as above and does the above inconsistency adjustment.
LINPRG	Solves one step, including resolution of blocking of the local linear programming problem. Determines the necessary pivot columns and rows.
TRAN	Called by LINPRG to do the actual pivot calculations. The pivot element is (LS,LT).
INCHR	(Interchange Rows.) Interchanges rows LR1 and LR2. Replaces LR2 in the control sequence by LR1.
INCHC	(Interchange Columns.) Interchanges rows LC1 and LC2 in the basis. Replaces column LC2 in the control sequence by column LC1.
BOUND	Does the calculations that preserve the bounds when doing the LINPRG procedure.
PRINT	Does the detailed column generation printouts when the intermediate print flag, IX1, is set to its maximum value of 4.
WRTOUT	Prints final solution and Lagrange multipliers. Terminates the run if the stop flag is set (if ISF $\geq$ 1).

INTERVAL DETERMINATION

Logic for procedure INVDET when all constraints are inequalities. This logic is repeated for row indices i from 1 through m - 1.

I. If $R_i = 0$.

A. If FL(i) = 0 done.

B. If FL(i) $\neq$ 0 set A1 = 0, A2 = 2.

i. If FL(i) > 0 set A1 = D - G1(i)/FL(i).

ii. If FL(i) < 0 set A2 = D - G1(i)/FL(i).

Go to step III.

II. If $R_i \neq 0$. Compute $DS1 = FL(i)^2 - 4 \cdot R_i[G1(i) - D]$.

i. Let $DS = \text{sign}(DS1) \cdot \sqrt{|DS1|}$.

ii. Let A1 = 0 and A2 = $+\infty$

and then do parts A, B, C, or D, depending on signs.

A. G1(i) > 0 and $R_i > 0$. If DS < 0 quit procedure, else

$$A1 = \frac{-FL(i) - DS}{2R_i} \quad \text{and} \quad A2 = \frac{-FL(i) + DS}{2R_i} .$$

B. G1(i) > 0 and $R_i < 0$. If DS < 0, done; else

$$A1 = \frac{-FL(i) - DS}{2R_i} .$$

C. G1(i) < 0 and $R_i > 0$.

$$A2 = \frac{1 - FL + DS}{2R_i} .$$

D. G1(i) < 0 and $R_i < 0$. If DS < 0 done, if FL(i) < 0 done; else

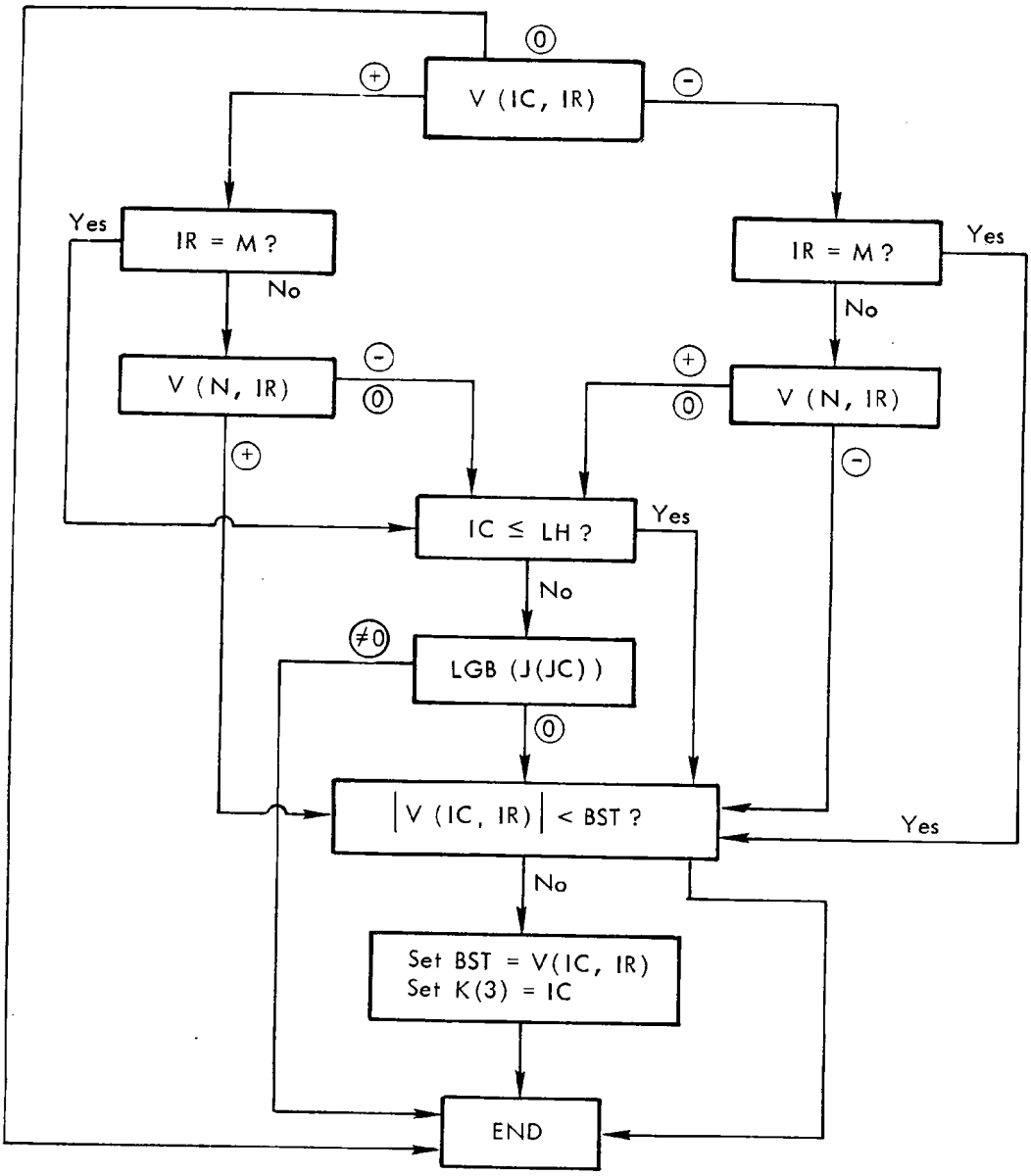
$$A2 = \frac{-FL(i) + DS}{2R_i} .$$

Now do step III.

III. A. If A2 ≤ ENPTR, set KFLG = 1.

B. Set ENPTL = max(A1, ENPTL)

ENPTR = min(A2, ENPTR).



Logic from range (30,39) of LINPRG. BST initially zero.
IC ranges from M1 + 1 to N - 1.

Fig. 12 — Logic of column selection when KO(3) is being determined

Appendix B
USE OF THE NONLINEAR PROGRAMMING ALGORITHM FOR OTHER APPLICATIONS

The PL/1 version of the nonlinear programming routine consists of twenty internal procedures enclosed by an external procedure named NONLIN. Of the twenty internal procedures, only four procedures need vary with the particular application. These four are SETUP, RESET, FCNGEN, and COLGEN. The function of each subprogram is quite simple, and it may be modified for other applications than the one presented here. These four procedures may, in fact, be removed from the main procedure and compiled separately, and the sixteen remaining procedures may be compiled into a load module that does not change. The global variables associated with the external procedure NONLIN must then be declared external, as these variables are used to pass information between the sixteen "fixed" procedures and the four varying procedures. A brief description of the four potentially varying procedures and an indication of how to use them follows.

In addition to communicating with the sixteen fixed procedures, the four varying subroutines must communicate with each other. In this application, these four procedures, together with the procedure "PAGE", have additional external variables defined in them. The definitions of these variables are as follows.

EXTERNAL VARIABLES FOR FOUR SUBROUTINES

Variable Name	Meaning
TITLE	Information in the title card.
B(·)	Right-hand side of linear constraints.
T(·)	The variable T_j.
SL(·)	The variable ℓ_j.
D(·)	The variable D_j.
CC(·)	The variable C_j.
VALUE(·)	The variable V_j.
PROB(·,·)	The variable P_{ij}.
PS(·)	The sum $\Sigma_i P_{ij} S_{ij}$.
TK(·)	An intermediate variable in the objective calculation, $\tilde{P}_j$ (Sec. II, Eq. (3.6)).
SMAX(·)	The maximum sorties for aircraft i, S_i.
ADC(·,·)	Coefficient matrix for the side constraints.
ROW(·)	Aircraft type associated with a side constraint.
NAT	Number of aircraft types.
NTT	Number of target types.
NADC	Number of side constraints.
NAATT	NAT + NTT.
IPAGE	Page number of last page printed.

The procedure SETUP does all of the data reading that is necessary to run the problem. If there are coefficients for the objective function or any of the constraints, these coefficients must be read in this procedure. The dimension of the problem, and any variable that is to be set to a nondefault value is computed in this procedure. In this application, SETUP reads in the information for T, D, CC, VALUE, PROB, SMAX, ADC, ROW, NAT, NTT, NADC, and NAATT. In addition it sets the value of M, the number of constraints plus one, to

$$\text{NAT + NTT + NADC + 1 ;}$$

and the value of N3, the number of independent variables, to

$$\text{NTT * NAT .}$$

Procedure SETUP also does any printing associated with the input.

The two procedures, FCNGEN and COLGEN, operate as a pair. FCNGEN evaluates the functions $g(y^o)$ and COLGEN evaluates the gradient $\nabla g(y^o)$. Note that the objective function is always the *mth* or last component of g. g and ∇g must be written correctly so that the program will behave consistently; if COLGEN and FCNGEN do not refer to the same functions, the main programs may behave erratically. There is a procedure for testing the consistency of COLGEN and FCNGEN; it is called "CONTEST" and is described in Appendix C.

FCNGEN is called with one parameter "JP". If JP = 1, then only $g^m(y^o)$ should be calculated. If JP = 0, then all m functions should be evaluated. The values of the calculated functions are put in the array "G"; i.e., G(M) is the objective value and G(1), G(2), ..., G(M - 1) are the values of the constraints.

The procedure COLGEN is also called with one parameter, called "JC". JC is the component or column number of the gradient to be evaluated. The procedure stores the *JCth* column of the gradient in CA(1), CA(2), ..., CA(M).

The procedure RESET is called when the nonlinear programming routines have determined that a termination condition has occurred.

RESET allows the user to print the output associated with the problem just completed, to input the data for a new problem as in SETUP, and to set the termination conditions. In this application, we only print the output and then set the termination condition by setting

$$ISF = 1 ,$$

where ISF, the "stop flag," is zero to continue and one to stop. In an application in which the basis is saved, ISF would be set to zero. If one wished to continue but with a new basis, the following statements should be included:

$$ISF = 0$$
$$LPC = 0,$$

where LPC is the parameter of the procedure.

Procedure RESET indicates whether the current solution is feasible. If K(2) is equal to M, the solution is feasible; otherwise it is not. The dual variables associated with the nonobjective rows may be recovered as X(·) as follows:

(1) Set X(·) = 0;

(2) For JC in the range [1, LH], set X[I(JC)] = V[JC, K(2)].

Note that the above representation of the V matrix is by column and row, which is the PL/1 convention and is opposite to the FORTRAN convention. When the problem is infeasible, the current objective row for the purpose of reducing infeasibility is I[K(2)].

Appendix C
THE CONTEST PROGRAM

The CONTEST program is a main procedure that is designed to evaluate new FCNGEN and COLGEN procedures. It will test these procedures for consistency and nonlinearity. (Any reasonably complicated set of functions is very likely to contain errors when the procedures are first written.)

In addition to FCNGEN and COLGEN, the procedure SETUP must be included in the CONTEST run. This is because SETUP initiates many of the parameters and sets the dimensions that are needed in the other procedures. In addition to checking the consistency of these routines, CONTEST will also detect compilation errors that might not be detected until the routines are run in the much larger general system.

To use CONTEST, the user sets up his input and writes SETUP, COLGEN, and FCNGEN as he would ordinarily do with the general system. Then he compiles these procedures along with the CONTEST main procedure (which is only about 93 cards), and runs with the normal "Compile and Go" job control language. Provided that the resulting program is error free, the output produced may then be checked for consistency.

For printed values of y, each line of output contains values for $g^i(y)$, $\nabla g^i(y_j)$, $\nabla_j g^i(y)$, ∇y^j, $FL(i) = \nabla_j g^i(y)\nabla y^j$, and $R(i) = \nabla g^i(y) - FL(i)$. If both $\nabla_j g^i(y)$ and $\nabla g^i(y_j)$ are zero, the above values are not printed for the corresponding (i,j) pairs. The program calculates a step-size ∇y^j that is one-tenth the difference between the upper and the lower bounds of each variable. Then $\nabla g^i(y_j) = g^i(y^o + \nabla y_j) - g^i(y^o)$. In turn, CA(i) as produced by COLGEN(j) is $\partial g^i(y)/\partial y^j$ or $\nabla_j g^i(y)$. When the factor g^i is linear, then $\nabla_j g^i(y)\nabla y_j = \nabla g^i(y_j)$, and R(i) is zero. When the function g^i is convex, then R(i) is positive. When the values printed by CONTEST check for function values, linearity, and convexity, then the procedures SETUP, COLGEN, and FCNGEN may be ready for testing in the general system.

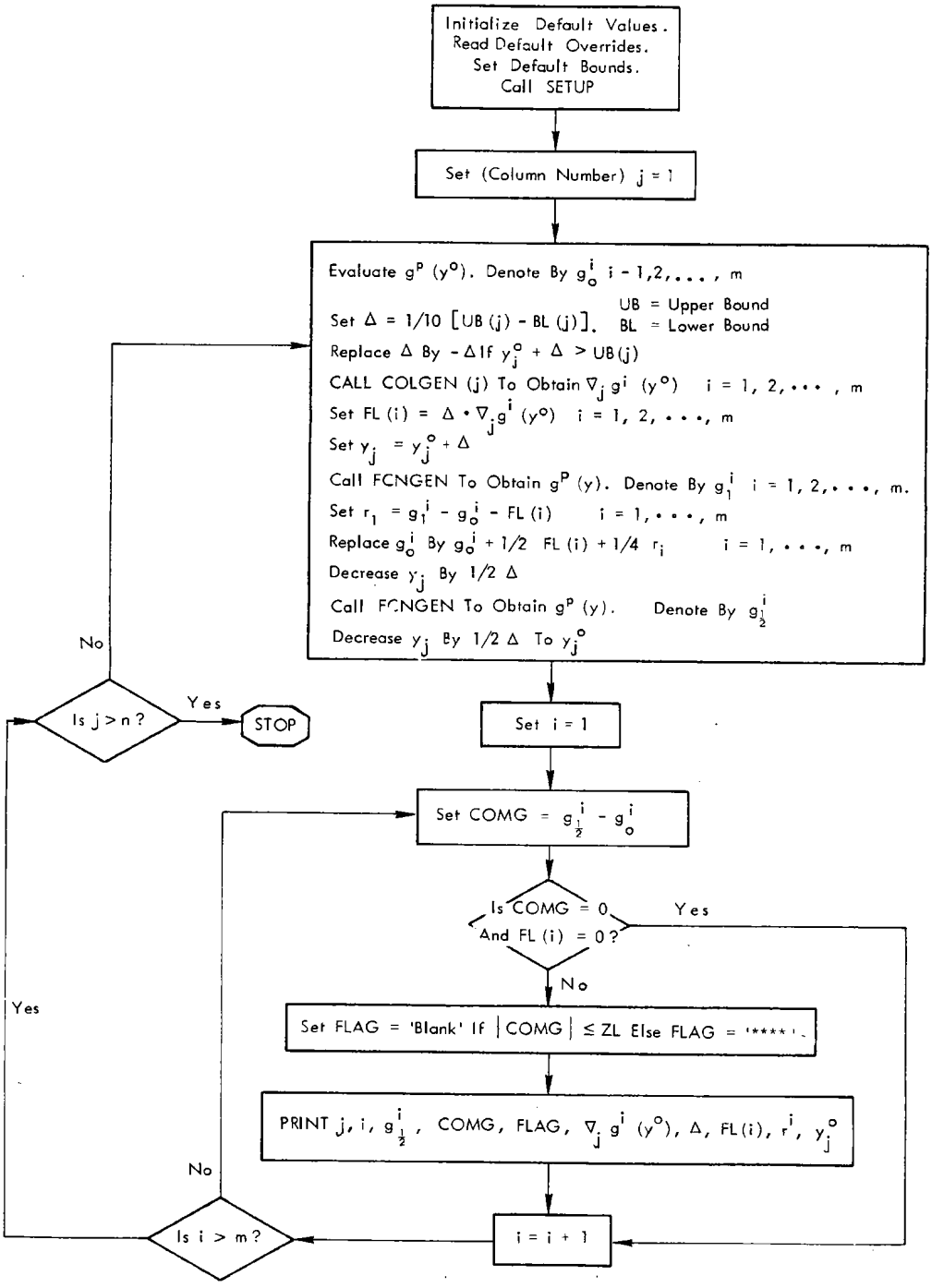


Fig. 13 — General flow of contest program

REFERENCES

1. Lu, J. Y., and R.B.S. Brooks, *WRM Requirements Computation for the Air Force Nonnuclear Air-to-Ground Munitions, Volume I: A Model*, The Rand Corporation, R-800/1-PR, October 1971.

2. Graves, G. W., "A Complete Constructive Algorithm for the General Mixed Linear Programming Problem," *Naval Research Logistics Quarterly, 12*, No. 1, March 1965, pp. 1-34.

3. -----, D. E. Pingry, and A. B. Winston, "Application of a Large-Scale Nonlinear Programming Problem to Pollution Control," in *AFIPS Conference Proceedings, 39*, AFIPS Press, 1972, pp. 123-134.

4. Graves, G. W., *Development and Testing of a Nonlinear Programming Algorithm*, Aerospace Corporation, Report No. ATR-64(7040)-2, June 1964.

5. -----, and A. B. Winston, "The Application of a Nonlinear Algorithm to a Second-Order Representation of the Problem," *Centre d'Etudes de Recherche Operationnelle, 11*, 1969, pp. 75-91.

6. Hatfield, G. B., and G. W. Graves, "Optimization of a Reverse Osmosis System Using Nonlinear Programming," *Desalinization, 7*, 1970, pp. 147-177.

7. Fiacco, A. V., and G. P. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and Sons, New York, 1968.

8. Greenstadt, J. L., "A Richocheting Gradient Method for Nonlinear Optimization," *J. SIAM Applied Mathematics, 14*, No. 3, 1966, pp. 429-445.

9. Rosen, J. B., "The Gradient Projection Method for Nonlinear Programming, Part II: Nonlinear Constraints," *J. SIAM, 9*, 1961, pp. 514-532.

10. Zoutendijk, G., *Methods of Feasible Directions*, American Elsevier, New York, 1960.

11. Buck, R. C., *Advanced Calculus*, McGraw Hill Book Company, New York, 1956.

12. Hadley, G., *Linear Programming*, Addison Wesley Publishing Company, Inc., Reading, Massachusetts, 1962.

13. Zangwill, W. I., *Nonlinear Programming*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.