MEMORANDUM
RM-4345-PR
JANUARY 1965

# THE NUMERICAL SOLUTION OF THE CHEMICAL EQUILIBRIUM PROBLEM

R. J. Clasen

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM
RM-4345-PR
JANUARY 1965

# THE NUMERICAL SOLUTION OF THE CHEMICAL EQUILIBRIUM PROBLEM

R. J. Clasen

This research is sponsored by the United States Air Force under Project RAND—Contract No. AF 49(638)-700 monitored by the Directorate of Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force.

The RAND Corporation
1700 MAIN ST • SANTA MONICA • CALIFORNIA • 90406

## PREFACE

This Memorandum is one in a continuing series of RAND publications dealing with theoretical computational questions arising from the RAND program of research in biology and physiology. The Memorandum contributes to our ability to apply computer technology to the analysis of complex chemical systems by considering the "chemical equilibrium problem," the problem of determining the distribution of chemical species that minimizes the free energy of a system while conserving the mass of each of the chemical elements.

Solutions to the chemical equilibrium problem published up to this time [4,5] apply to those problems for which an estimate of the solution exists. This Memorandum considers a problem for which no estimated solution exists and solves that problem with the maximum precision now available.

The mathematical aspects of this Memorandum should also be of interest in other fields where computational analyses of complex chemical systems are under consideration, e.g., in studies of rocket propulsion systems, planetary atmospheres, re-entry problems, etc.

## FOREWORD

In deciding between the languages of mathematics and physical chemistry, we have chosen in this Memorandum to use that of mathematics. The disadvantage of this choice is that the physical chemist may experience some difficulty in immediately identifying certain concepts. The advantage is that mathematical language divorces the methods from the physical assumptions involved in constructing a mathematical model of a physical system.* The mathematical methods are, hence, free to transcend their specific chemical applications.

The methods given here do not solve every problem that is specified in the given mathematical form. The solution of a problem in which some phase vanishes (a degenerate problem) requires further work. Some work has been done on particular degenerate systems [13], but the accurate numerical solution of a large general system of this type has yet to be accomplished. Until recently, a skilled physical chemist could intuitively eliminate the degeneracies of his model and

*The reader is referred to other works for the procedure of constructing the mathematical models of biochemical systems [9-12].

obviate the need for solving a degenerate system. But, as problems grow, eliminating degeneracy becomes increasingly difficult. Frequently, the point at which the problem becomes too large for the physical chemist to decide whether or not to include a phase coincides with the point at which the problem becomes numerically unwieldy. Hopefully, the future will eliminate these difficulties.

Statements about convergence and convergence tests exist, unless otherwise indicated, in the context of finite-accuracy numerics. Statements of this kind do not mean, in the absence of qualification, that no problem exists nor that no machine would serve as a counter example. Rather they are simply descriptions of what was found to occur in actual practice.

No attempt has been made to describe those methods which were tried and found wanting. The methods presented are those which are best for the largest number of cases.

Finally, it should be pointed out that although computing time was a factor, it was considered secondary to accuracy of results.

## ACKNOWLEDGMENTS

The author wishes to thank J. C. DeHaven, E. C. DeLand, F. R. Gilmore, and N. Z. Shapiro for their many constructive comments and suggestions.

## CONTENTS

## 1. INTRODUCTION

For the purposes of this Memorandum, the chemical equilibrium problem is merely a name we use for a particular mathematical programming problem, i.e., the problem of minimizing a particular nonlinear function $F(x_1,x_2,\ldots,x_n)$, defined below, while satisfying the linear restraints or constraints

$$\sum_{j=1}^{n} a_{ij}\, x_j = b_i \qquad i=1,2,3,\ldots,m \qquad (1.1)$$

with $x_j \geq 0$ for $j=1,2,\ldots,n$ and $a_{ij}$, $b_i$ given constants. Assuming that the equations of (1.1) are linearly independent, then in order to have a non-trivial problem it can be assumed that $m<n$. The variables $x_1,x_2,\ldots,x_n$ can be considered components of a vector $(x_1,x_2,\ldots,x_n)$. Solving the chemical equilibrium problem then is the problem of determining this vector. The variable $x_j$ will be referred to as the "$j^{th}$ component"; also the numerical value of $x_j$ may be referred to as the "component" rather than using the perhaps linguistically correct but cumbersome term "component value."

The components are partitioned into p non-empty subsets called compartments. Let us denote these compartments by $\langle 1\rangle, \langle 2\rangle, \ldots, \langle p\rangle$. Then if the $j^{th}$ component is in the $k^{th}$ compartment, we will say $j \epsilon \langle k\rangle$, where each component is in exactly one compartment. The number of the compartment that the $j^{th}$ component is in is denoted by $[j]$. Hence $j \epsilon \langle k\rangle$ implies $[j] = k$, and conversely. Each compartment has associated with it a sum defined by

$$S_k = \sum_{j \epsilon \langle k\rangle} x_j \ . \tag{1.2}$$

The component fraction $\hat{x}_j$ is defined by $\hat{x}_j = \frac{x_j}{S_{[j]}}$ whenever $S_{[j]} > 0$.

The objective function to be minimized over (1.1) is

$$F(x_1, x_2, \ldots, x_n) = \sum_{j=1}^{n} x_j(c_j + \log \hat{x}_j) \tag{1.3}$$

where $c_1, c_2, \ldots, c_n$ are given constants, called objective constants.

When an $x_j$ is zero, $\log \hat{x}_j$ is undefined; but we define 0 log 0 to equal 0 so that we may evaluate F when some components are zero. A *feasible solution* to the chemical equilibrium problem is defined to be any set of *non-negative* components that satisfies (1.1). The problem is said to be *feasible* if it has feasible solutions. If no feasible solution is arbitrarily large in any component, the feasible problem is said to be *bounded feasible*; all practical problems with which one might have occasion to deal are bounded feasible.

A *solution* or *optimal solution* to a bounded feasible problem is any feasible solution in which $F(x_1,\ldots,x_n)$ attains the minimum value over all feasible solutions. A problem which has optimal solutions in which some component is zero is called *degenerate*, and a bounded feasible problem in which the components in any optimal solution are all strictly positive is called a *non-degenerate* problem. It has been shown [1, Theorem 12.1] that a non-degenerate problem has exactly one optimal solution. Hence, we may speak of *the* solution to the problem. Furthermore, it has also been shown* for the non-degenerate problem that the minimization of F is equivalent to the existence of numbers $\pi_1,\pi_2,\ldots,\pi_m$, called Lagrange multipliers, which satisfy:

*Ref. 1, p. 18.

$$\sum_{i=1}^{m = \#\ \text{rows}} \pi_i a_{ij} = c_j + \log \hat{x}_j \ . \qquad j=1,2,3,\ldots,n \qquad (1.4)$$

In the following sections we derive conditions, analogous to (1.4), which are useful in solving the problem. In Sec. 2 we are interested in finding a solution to (1.1) with all $x_j > 0$. A set of $x_j$ which satisfies these conditions is called a positive feasible solution. If (1.1) is satisfied with $x_j \geq 0$, we have called such a result a feasible solution. The theory of linear programming gives us methods of finding feasible solutions to problems with linear restraints. In Sec. 2, we use a linear programming technique to find a positive feasible solution. In Sec. 4 we show how to modify the initial positive feasible solution to get the solution to the problem.

## 2. THE INITIAL SOLUTION

The algorithms presented in the following sections require an initial positive feasible solution in order that the procedure for solving the problem can be initiated. Frequently, an individual with a problem to solve will be able to give a rather accurate estimate of its optimal solution. This estimate may be the exact solution of another problem which differs from the one being considered in relatively minor ways.

### THE PROJECTION METHOD

Let us suppose that such is the case, and let us denote the estimate of the components by $y_1, y_2, \ldots, y_n$. These values, substituting $y_j$ for $x_j$ in Eq. (1.1), will not generally satisfy (1.1), being somewhat in error. Let us denote these errors by $g_1, g_2, \ldots, g_m$; that is, let

$$g_i = b_i - \sum_{j=1}^{n} a_{ij} y_j \ . \qquad i=1,2,\ldots,m \qquad (2.1)$$

Then, we wish to find corrections to $y_j$ such that, denoting the corrections by $\theta_j$, we have

$$b_i - \sum_{j=1}^{n} a_{ij}(y_j + \theta_j) = 0 \quad i=1,2,\ldots,m$$

or

$$g_i = \sum_{j=1}^{n} a_{ij}\theta_j \ . \qquad i=1,2,\ldots,m \qquad (2.2)$$

The $\theta_j$ must also be chosen such that $y_j + \theta_j > 0$, for all j. We cannot guarantee this condition, but we can attempt to choose small values for $\theta_j$. One way to do this is to minimize

$$\sum_{j=1}^{n} w_j\theta_j^2$$

subject to (2.2), where $w_j$ is the "weight" or relative importance of minimizing $\theta_j$. This reduces to the problem of finding Lagrange multipliers $\pi_1,\pi_2,\ldots,\pi_m$, such that with

$$L = \frac{1}{2} \sum_{j=1}^{n} w_j\theta_j^2 - \sum_{i=1}^{m} \pi_i \left( \sum_{j=1}^{n} a_{ij}\theta_j - g_i \right) \qquad (2.3)$$

we have

$$\frac{\partial L}{\partial \theta_j} = 0 \ . \qquad j=1,2,\ldots,n \qquad (2.4)$$

Equation (2.4) becomes

$$w_j\theta_j = \sum_{i=1}^{m} a_{ij}\pi_i \qquad j=1,2,\ldots,n \qquad (2.5)$$

and substituting (2.5) into (2.2) we have

$$g_i = \sum_{\ell=1}^{m}\left[\pi_\ell\left(\sum_{j=1}^{n}\frac{a_{\ell j}a_{ij}}{w_j}\right)\right]. \quad i=1,2,\ldots,m \qquad (2.6)$$

The terms

$$\sum_{j=1}^{n}\frac{a_{\ell j}a_{ij}}{w_j}$$

can be immediately evaluated; let us denote these terms by

$$q_{\ell i} = \sum_{j=1}^{n}\frac{a_{\ell j}a_{ij}}{w_j} \quad . \qquad (2.7)$$

Note that $q_{\ell i} = q_{i\ell}$. Then, (2.6) becomes

$$g_i = \sum_{\ell=1}^{m} q_{\ell i}\pi_\ell \quad . \qquad i=1,2,\ldots,m \qquad (2.8)$$

Equation (2.8) is a set of m simultaneous equations in the m unknowns, $\pi_1,\pi_2,\ldots,\pi_m$. These equations may be solved for $\pi_1,\pi_2,\ldots,\pi_m$, and then these values may be substituted in (2.5) to get $\theta_1,\theta_2,\ldots,\theta_n$. There remains the question of choosing values for the weighting factors $w_j$. In tests of this method, it has been found that using

$$w_j = \frac{1}{y_j}$$

yields satisfactory results. The choice of the weighting factors depends, to some extent, on the available computers. Using these weighting factors, we can summarize the computation of $\theta_j$ in the following three equations:

$$q_{\ell i} = \sum_{j=1}^{n} a_{\ell j} a_{ij} y_j \qquad \begin{matrix} i=1,2,\ldots,m \\ \ell=1,2,\ldots,m \end{matrix} \qquad (2.9)$$

$$\sum_{\ell=1}^{m} q_{\ell i}\pi_\ell = b_i - \sum_{j=1}^{n} a_{ij} y_j \qquad i=1,2,\ldots,m \qquad (2.10)$$

$$\theta_j = y_j \sum_{i=1}^{m} a_{ij}\pi_i \qquad j=1,2,\ldots,n \qquad (2.11)$$

where

$$x_j = y_j + \theta_j \quad . \qquad j=1,2,\ldots,n \qquad (2.12)$$

The $x_j$ from (2.12) will satisfy (1.1). However, the $x_j$ need not all be strictly positive. If any $x_j$ is zero or negative, this method of obtaining the initial solution, which we shall call the projection method, has failed. If the projection method fails, or if no initial estimate is provided, then a linear programming method may be used.

## THE LINEAR PROGRAMMING METHOD

The terminology used in linear programming is similar to the terminology used above in describing the chemical equilibrium problem. The statement of a linear programming problem includes a set of linear restraints

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad i=1,2,\ldots,m \qquad (2.13)$$

together with a set of constants $C_1, C_2, C_3, \ldots, C_n$, called costs. A feasible solution to a linear programming problem is any set of non-negative $x_j$ such that (2.13) is satisfied. The costs are used to form the following expression, L, which is called the objective function

$$L = \sum_{j=1}^{n} C_j x_j \quad . \tag{2.14}$$

For every set of feasible $x_j$, we can evaluate L. The set of feasible $x_j$ for which L has the minimum value that it can have with any set of feasible $x_j$, is called a solution of the linear programming problem. A problem which has sets of feasible $x_j$ is called a feasible problem, and a problem in which there are no sets of feasible $x_j$ is called an infeasible problem. An infeasible problem has no solutions, while a feasible problem has at least one solution. In this discussion, we will not be concerned as to whether a problem has more than one solution: we will only be concerned with finding a solution to the problem. Since the means of finding a solution to a linear programming problem has been the subject of many papers and books, we will not give an actual method of solving the linear programming problem here. The reader may refer to Dantzig [2] for a complete discussion of the problem.

The problem of finding a feasible solution to a linear programming problem is itself a linear programming problem--that is, it involves finding a solution to the problem with all $C_j$ equal to zero. With all $C_j = 0$, L in (2.14) is zero for any set of feasible $x_j$; hence, L is at its minimum value for any set of feasible $x_j$. Since L is at its minimum value for any feasible set of $x_j$, any feasible set of $x_j$ is, by the above definition, a solution to the linear programming problem.

However, we must not only find a feasible solution to the linear programming problem, we must also find a positive feasible solution to the problem. In order to do this, we let

$$x_j = y_j + y_{n+1} \quad . \qquad j=1,2,\ldots,n \qquad (2.15)$$

If we can find non-negative values of $y_1, y_2, \ldots, y_{n+1}$ which satisfy

$$\sum_{j=1}^{n} a_{ij}(y_j + y_{n+1}) = b_i \qquad i=1,2,\ldots,m \qquad (2.16)$$

then $x_j$, as defined by (2.15), will be a feasible solution. If we can somehow assure that $y_{n+1}$ is positive, then all $x_j$ will be positive. Rewriting (2.16), we have

$$\sum_{j=1}^{n} a_{ij}y_j + \left(\sum_{j=1}^{n} a_{ij}\right) y_{n+1} = b_i \ . \qquad i=1,2,\ldots,m \qquad (2.17)$$

If we now specify $C_1, C_2, \ldots, C_{n+1}$, we have a linear programming problem in n+1 unknowns. In order to guarantee that $y_{n+1}$ is positive, if it is possible for it to be positive, we can <u>maximize</u> $y_{n+1}$. It is easy to see that we can maximize $y_{n+1}$ by setting

$$L = - y_{n+1} \qquad (2.18)$$

which is equivalent to setting $C_1=C_2=C_3=\ldots=C_n=0$, $C_{n+1} = -1$. If the solution to the resulting linear programming problem is feasible and $y_{n+1} > 0$, then we have, by (2.15), a positive feasible solution to the analogous chemical equilibrium problem (1.1). If the linear programming problem is feasible but $y_{n+1} = 0$, then the analogous chemical equilibrium problem is degenerate, since there is no strictly positive solution to the problem. However, this is a rather trivial kind of degeneracy, and its occurrence usually indicates that a mistake was made in setting up the problem. Hence, this linear programming method gives us a way of finding a positive feasible solution to the chemical equilibrium problem if the chemical equilibrium problem is non-degenerate.

The positive feasible solution that we obtain by this method will generally not resemble the final solution of the chemical equilibrium problem. The initial positive feasible solution can be improved by the following technique. Define $b_{m+1}$ to be some multiple, between zero and one, of the value of $y_{n+1}$ that was obtained above. Then, adjoin to the linear restraints (2.17) one more restraint of the form $y_{n+1} = b_{m+1}$. Next, solve the linear programming problem with these restraints and with $C_1=c_1$, $C_2=c_2$, ..., $C_n=c_n$, $C_{n+1}=0$ (recall that the lower-case c's here refer to the c's in the chemical equilibrium problem (1.3)). The solution to this linear programming problem will give a set of components more nearly resembling the solution to the chemical equilibrium problem than did the components calculated from Eqs. (2.17) and (2.18). This new solution, in turn, may be improved by solving another linear programming problem (the details of which can be seen in SUBROUTINE LP in Appendix A) and averaging the new solution with the old solution.

In order to solve an elaborate chemical equilibrium problem it is not sufficient to simply use a method which we can prove converges to the correct solution. Proofs of convergence generally assume infinite computational accuracy, but since we are usually limited in practice to about eight significant digits, the numerical solution will not always converge. However, it has been observed that the closer we can get to the solution by the initial solution methods described above, the greater will be the probability that the numerical procedure will converge. Furthermore, not only will the probability of convergence be greater, but the number of iterations to get to the solution will be fewer, and hence--when an improved initial solution is used--the computation time will be shorter. Unfortunately, the mathematical methods that are available for analyzing convergence of iterative processes do not, in the case of the chemical equilibrium problem, enable us to prove convergence when we are limited to finite mathematical accuracy. Only experience with a particular method will tell us whether it is a useful numerical procedure to use.

In the next section we consider a somewhat more general problem than the chemical equilibrium problem. This problem is considered first because the numerical results take on an especially simple form when the additional generality is admitted.

## 3. THE LINEAR-LOGARITHMIC PROGRAMMING PROBLEM, FIRST-ORDER METHOD

In this section we consider the problem of minimizing

$$F(x_1,x_2,\ldots,x_N) = \sum_{j=1}^{N} x_j(c_j + d_j \log x_j) \qquad (3.1)$$

while satisfying the linear restraints

$$\sum_{j=1}^{N} a_{ij}x_j = b_i \ . \qquad i=1,2,3,\ldots,M \qquad (3.2)$$

The symbols $a_{ij}$, $b_i$, $c_j$, and $d_j$ denote constants, and $x_1,x_2,\ldots,x_N$ are the unknowns that we seek. We restrict the problem to the case that $d_j \neq 0$ for $j = 1,2,3,\ldots,N$. We note that if $x_j < 0$, the term in (3.1), $x_j(c_j + d_j\log x_j)$, is undefined, whereas if $x_j > 0$ this term is defined. If $x_j = 0$ we define $x_j(c_j + d_j\log x_j) = 0$, since this expression approaches zero as $x_j > 0$ approaches zero. From this discussion, we see that, in order for a solution of Eqs. (3.1) and (3.2) to be defined, we must assume that $x_j \geq 0$ for $j = 1,2,3,\ldots,N$.

We may attempt to solve this problem using Lagrange multipliers.[*] In this method we let

$$L = F(x_1,x_2,x_3,\ldots,x_N) - \sum_{i=1}^{M}\left[\pi_i\left(\sum_{j=1}^{N} a_{ij}x_j - b_i\right)\right]$$

and then set

$$\frac{\partial L}{\partial x_j} = 0$$

for j = 1,2,3,...,N. Performing the partial differentiation, we get

$$c_j + d_j \log x_j + d_j - \sum_{i=1}^{M} \pi_i a_{ij} = 0 \;, \qquad (3.3)$$

$$j=1,2,3,\ldots,N$$

or, when rearranged,

$$\log x_j = d_j^{-1}\left[\sum_{i=1}^{M} \pi_i a_{ij} - c_j - d_j\right] . \qquad (3.4)$$

$$j=1,2,3,\ldots,N$$

[*] See Kaplan, Ref. 3, p. 128, or Dantzig, Ref. 2, p. 140.

Exponentiating both sides of (3.4), we get

$$x_j = \exp\left[ d_j^{-1} \sum_{i=1}^{M} \pi_i a_{ij}^{\alpha} - d_j^{-1} c_j - 1 \right] . \qquad (3.5)$$

$$j=1,2,3,\ldots,N$$

Note that for (3.5) to be a solution to the problem, we must have all $x_j > 0$. We assume, in the remainder of this section, that the solution does have all $x_j > 0$. Then, the problem reduces to the problem of determining the M $\pi_i$ so that the $x_j$ from (3.5) satisfy (3.2) Equivalently, the M + N equations (3.2) and (3.5) must be satisfied simultaneously by the proper choice of the M + N unknowns, $\pi_1, \pi_2, \ldots, \pi_M$, $x_1, x_2, \ldots, x_N$. We now consider two methods of approximating the solution.

In the first method, we suppose that we have an estimate of the $x_j$ which may or may not satisfy (3.2). We denote this estimate by $y_j$, and, in this method, solve Eqs. (3.2) and (3.4) simultaneously by making a linear approximation to log $x_j$. Since we have the estimate that $x_j$ is near $y_j$, we note that the first-order Taylor expansion of log $x_j$ about $y_j$ is

$$\log x_j = \log y_j + \frac{x_j - y_j}{y_j} + \text{(higher-order terms)} \ . \qquad (3.6)$$

Dropping the higher-order terms, and substituting (3.6) into (3.4) and solving for $x_j$, we have

$$x_j = y_j \left[ d_j^{-1} \sum_{i=1}^{M} \pi_i a_{ij}^{\alpha} - d_j^{-1} c_j - \log y_j \right] . \qquad (3.7)$$

$$j=1,2,3,\ldots,N$$

Now, if we substitute these $x_j$ into (3.2), we get

$$\sum_{\ell=1}^{M} \left( \sum_{j=1}^{N} d_j^{-1} a_{ij}^{\alpha} a_{\ell j}^{\beta} y_j \right) \pi_\ell = b_i + \sum_{j=1}^{N} a_{ij}^{\beta} y_j (\log y_j + d_j^{-1} c_j) \ .$$

$$i=1,2,3,\ldots,M$$

Denoting

$$r_{i\ell} = \sum_{j=1}^{N} d_j^{-1} a_{ij}^{\alpha} a_{\ell j}^{\beta} y_j \qquad \begin{matrix} \ell=1,2,3,\ldots,M \\ i=1,2,3,\ldots,M \end{matrix} \qquad (3,8)$$

and

$$s_i = b_i + \sum_{j=1}^{N} a_{ij}^{\beta} y_j (\log y_j + d_j^{-1} c_j) \qquad (3.9)$$

$$i=1,2,3,\ldots,M$$

we have

$$\sum_{\ell=1}^{M} r_{i\ell}\pi_\ell = s_i \ . \qquad i=1,2,3,\ldots,M \qquad (3.10)$$

Equation (3.10) is a set of simultaneous equations which can be solved for $\pi_1,\pi_2,\ldots,\pi_M$.

With the above results, we can now define the iterative process for the first method. At each iteration we have a set of values for $x_1,x_2,\ldots,x_N$. At the beginning of the iteration these values are called $y_1,y_2,\ldots,y_N$, and at the end of the iteration the values are $x_1,x_2,\ldots,x_N$. If

$$\frac{x_j-y_j}{y_j}$$

is small for each j, then we say we have converged. The magnitude of "small" depends on the nature of the problem. If

$$\frac{x_j-y_j}{y_j}$$

is not small for some j, then we have not converged and the iteration must be repeated. One iteration consists of the following three steps:

1) Evaluate terms in Eqs. (3.8) and (3.9), these terms depending on $y_1, y_2, \ldots, y_N$;

2) Solve Eq. (3.10) for $\pi_1, \pi_2, \ldots, \pi_M$;

3) Substitute $\pi_1, \pi_2, \ldots, \pi_M$ into (3.7) to get $x_1, x_2, \ldots, x_N$.

For this problem, in this generality, we can say nothing about whether this iterative process converges. In the next section we will show that the chemical equilibrium problem is a special case of this problem, and one for which, with appropriate modification, this method does converge.

## 4. THE FIRST-ORDER METHOD FOR SOLVING THE CHEMICAL EQUILIBRIUM PROBLEM

The chemical equilibrium problem is a special case of the linear-logarithmic programming problem. In order to put Eqs. (3.1) and (3.2) into the form of Eqs. (1.1) and (1.3), we first define

$$N = n+p$$

$$M = m+p$$

where, as stated previously, p is the number of compartments in the problem. Then we define $a_{ij}$, $b_i$, $x_j$, and $c_j$, for $i > m$ and $j > n$, as follows

$$b_i = 0 \qquad i=m+1,m+2,\ldots,M \qquad (4.1)$$

$$c_j = 0 \qquad j=n+1,n+2,\ldots,N \qquad (4.2)$$

$$x_{k+n} = S_k \qquad k=1,2,\ldots,p \qquad (4.3)$$

$$a_{ij} = \begin{cases} 0 & \text{if} \quad i \le m,\ j > n \\ 1 & \text{if} \quad i > m,\ j \le n, \text{ and } [j] = i-m \\ 0 & \text{if} \quad i > m,\ j \le n, \text{ and } [j] \neq i-m \\ -1 & \text{if} \quad i > m,\ j > n, \text{ and } i-m = j-n \\ 0 & \text{if} \quad i > m,\ j > n, \text{ and } i-m \neq j-n \ . \end{cases} \tag{4.4}$$

For all j, we define

$$d_j = \begin{cases} +1 & \text{if} \quad j \le n \\ -1 & \text{if} \quad j > n \ . \end{cases} \tag{4.5}$$

With these definitions, it has been shown [4] that the two problems are identical. Next, we let

$$x_j = y_j + \theta_j \tag{4.6}$$

$$\pi_i = \begin{cases} \pi_i' & i \le m \\ \pi_i' + \log S_{i-m} + 1 \ . & i > m \end{cases}$$

Substituting Eqs. (4.1) through (4.6) into (3.7) through (3.10) and simplifying, we have

$$\theta_j = y_j \left[ \sum_{i=1}^{m} a_{ij}\pi_i' - c_j - \log \hat{y}_j + \pi_{[j]+m}' \right] \qquad j=1,2,\ldots,n \tag{4.7}$$

$$r_{i\ell} = \begin{cases} \sum\limits_{j=1}^{n} \overset{\beta}{a}_{ij}\overset{\alpha}{a}_{\ell j} y_j & \ell \le m,\ i \le m \\ \sum\limits_{j\in\langle i-m\rangle} a_{\ell j} y_j & \ell \le m,\ i > m \\ \sum\limits_{j\in\langle \ell-m\rangle} a_{ij} y_j & \ell > m,\ i \le m \\ 0 & \ell > m,\ i > m \end{cases} \tag{4.8}$$

$$s_i' = \begin{cases} b_i + \sum\limits_{j=1}^{n} a_{ij} y_j (c_j + \log \hat{y}_j - 1) & i \le m \\ \sum\limits_{j\in\langle i-m\rangle} y_j (c_j + \log \hat{y}_j) & i > m \end{cases} \tag{4.9}$$

$$\sum_{\ell=1}^{M} r_{i\ell}\pi_i' = s_i' . \qquad i=1,2,\ldots,M \tag{4.10}$$

The directional derivative of F in the direction $(\theta_1,\theta_2,\ldots,\theta_n)$ is given by [1, Theorem 8.11] to be

$$\sum_{j=1}^{n} \theta_j(c_j + \log \hat{y}_j) \ . \tag{4.11}$$

But, if we compute $\sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j}$ where by (3.7)

$$\theta_{k+n} = S_k \left[\pi_{m+k} - \log S_k - 1\right] = S_k \, \pi'_{m+k} \qquad k=1,2,\ldots,p \tag{4.12}$$

we show, in Appendix B, that

$$\sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j} = - \sum_{j=1}^{n} \theta_j(c_j + \log \hat{y}_j) + \sum_{i=1}^{m} \pi_i \left(b_i - \sum_{j=1}^{n} a_{ij} y_j\right) . \tag{4.13}$$

Thus, if we assume that $(y_1, y_2, \ldots, y_n)$ is feasible, we get the interesting result that the directional derivative of F in the direction $(\theta_1, \theta_2, \ldots, \theta_n)$ is

$$\sum_{j=1}^{n} \theta_j(c_j + \log \hat{y}_j) = - \sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j} \leq 0 \ . \tag{4.14}$$

However, it is also shown in Appendix B that the equality on the right side of (4.14) holds if and only if the values for $y_j$ are optimal. We further note that if $(y_1, y_2, \ldots, y_n)$ is feasible, then

$$\sum_{j=1}^{n} a_{ij}\theta_j = 0$$

for i = 1,2,...,m. Hence, if $(y_1,y_2,\ldots,y_n)$ is feasible, then $(y_1+\lambda\theta_1,y_2+\lambda\theta_2,\ldots,y_n+\lambda\theta_n)$ will be feasible for any $\lambda$ for which each $y_j + \lambda\theta_j$ is positive.

We now state the first-order chemical equilibrium algorithm:

1) Calculate $(\theta_1,\theta_2,\ldots,\theta_n)$ using Eqs. (4.7) through (4.10).

2) Calculate the directional derivative of F in the direction $(\theta_1,\theta_2,\ldots,\theta_n)$ as given by Eq. (4.11); if this quantity is not negative, we are done.

3) Calculate

$$\epsilon = \sqrt{\frac{1}{n} \sum_{j=1}^{n} \left(\frac{\theta_j}{y_j}\right)^2} .$$

$\epsilon$ is a number that represents the root-mean-square error in $(y_1,y_2,\ldots,y_n)$. If $\epsilon$ is less than some given number (say, 0.001), we are done.

4) Calculate the ratio $-y_j/\theta_j$ for every j for which $\theta_j < 0$. Let $\lambda_1$ be the minimum of all such ratios and let $\lambda = \min(1, \beta\lambda_1)$, where $\beta$ is a number less than 1 but close to 1 (say, 0.99). We now perform the following steps until the test at c) below is satisfied:

a) Let $z_j = y_j + \lambda\theta_j$;

b) Compute the directional derivative of F at $z_j$ in the direction $(\theta_1, \theta_2, \ldots, \theta_n)$: $f(\lambda) = \theta_j(c_j + \log \hat{z}_j)$;

c) If $f(\lambda) \leq 0$, go directly to step 5);

d) Replace $\lambda$ by $\gamma\lambda$, where $0 < \gamma < 1$, e.g., $\gamma = \frac{1}{2}\sqrt{2}$.

5) Finally, replace $y_j$ by $y_j + \lambda\theta_j$ for $j = 1, 2, \ldots, n$.

Steps 1-5 are repeated until either the test in step 2 or the test in step 3 is satisfied.

If this process terminates, the solution will be optimal within the specified limits of accuracy. It may happen that the process does not terminate. Since the objective function F is convex* and assuming infinite computational accuracy, non-termination can occur only because the values chosen for $\lambda$ become smaller on every

*Ref. 1, Theorem 8.13; Ref. 5.

iteration. This will occur only if some $y_j$ is approaching zero, and hence $(y_1, y_2, \ldots, y_n)$ is approaching a point at which, if it were the optimal solution, the problem would be degenerate. It is possible for this to happen for a non-degenerate problem for which the initial solution chosen was too far from the optimal solution. Convergence can be guaranteed by imposing the condition that the value of F at the initial solution be less than the value of F at any feasible, degenerate point. However, it is not practical to impose this condition on the initial solution since it may be very difficult to find such a point. In practice, it has been found that round-off errors cause more difficulty than the possible selection of a poor initial solution.

## 5. THE LINEAR-LOGARITHMIC PROGRAMMING PROBLEM, SECOND-ORDER METHOD

In the first-order method, presented in Sec. 3, the iterative process was initiated with an estimate of the value of $x_1, x_2, \ldots, x_N$. In the second-order method, we assume that the problem is as defined by Eqs. (3.1) and (3.2), but that we have initial estimates for the values of $\pi_1, \pi_2, \ldots, \pi_M$. Let us denote these estimates by $\lambda_1, \lambda_2, \ldots, \lambda_M$. The $x_j$ can then be evaluated by Eq. (3.5), substituting $\lambda_i$ for $\pi_i$. These $x_j$, however, probably will not satisfy Eq. (3.2). The problem of the second-order method is to find numbers $\Delta\lambda_1, \Delta\lambda_2, \ldots, \Delta\lambda_M$, such that

$$\pi_i = \lambda_i + \Delta\lambda_i \qquad i=1,2,\ldots,M \qquad (5.1)$$

when substituted into (3.5) will give $x_j$ that satisfy (3.2).

In order to accomplish this, we first use the $x_j$ calculated from Eq. (3.5) to get

$$g_i = b_i - \sum_{j=1}^{N} a_{ij} x_j \qquad i=1,2,\ldots,M \qquad (5.2)$$

where $g_i$ represents the amount that equation i is in error. Next, we evaluate

$$\frac{\partial g_i}{\partial \lambda_\ell}$$

by

$$\frac{\partial g_i}{\partial \lambda_\ell} = \frac{\partial}{\partial \lambda_\ell}\left[b_i - \sum_{j=1}^{N} a_{ij}^{\beta} x_j\right] = - \sum_{j=1}^{N} a_{ij}^{\beta} \frac{\partial x_j}{\partial \lambda_\ell}$$

$$= - \sum_{j=1}^{N} a_{ij}^{\beta} \frac{\partial}{\partial \lambda_\ell}\left[\exp\left(d_j^{-1} \sum_{h=1}^{M} \lambda_h a_{hj}^{\alpha} - d_j^{-1} c_j - 1\right)\right]$$

$$= - \sum_{j=1}^{N} a_{ij}^{\beta} d_j^{-1} x_j a_{\ell j}^{\alpha} = - r_{\ell i} \qquad (5.3)$$

where $r_{\ell i}$ is given by Eq. (3.8). If we make a very small change, $d\lambda_1$, $d\lambda_2, \ldots,$ in $\lambda_1, \lambda_2, \ldots,$ the change in $g_1, g_2, \ldots,$ is given by $dg_1, dg_2, \ldots,$ where

$$dg_i = + \sum_{\ell=1}^{M} \frac{\partial g_i}{\partial \lambda_\ell} d\lambda_\ell \qquad i=1,2,\ldots,M$$

or

$$dg_i = -\sum_{\ell=1}^{M} r_{\ell i} d\lambda_\ell \; . \qquad i=1,2,\ldots,M \qquad (5.4)$$

We would want $dg_i$ to be equal to $-g_i$ as computed by Eq. (5.2). If we make the approximation that

$$\frac{\partial g_i}{\partial \lambda_\ell}$$

is constant over the domain considered, we can set $dg_i = -g_i$, let $d\lambda_\ell = \Delta\lambda_\ell$, and write

$$g_i = \sum_{\ell=1}^{M} r_{\ell i} \Delta\lambda_1 \; . \qquad i=1,2,\ldots,M \qquad (5.5)$$

Equation (5.5) consists of M equations in the M unknowns $\Delta\lambda_1, \Delta\lambda_2, \ldots, \Delta\lambda_M$. We may thus solve Eq. (5.5) for $\Delta\lambda_1, \Delta\lambda_2, \ldots, \Delta\lambda_M$ and compute $\pi_1, \pi_2, \ldots, \pi_M$ from (5.1). If the assumption about

$$\frac{\partial g_i}{\partial \lambda_\ell}$$

being constant over the domain considered was correct, then the $x_j$ computed from (3.5) with these values for $\pi_i$ will satisfy (3.2). However, in general, they will not satisfy (3.2), but, if we were close enough to the solution so that the

$$\frac{\partial g_i}{\partial \lambda_\ell}$$

did not vary greatly in the domain considered, then the new values for $x_j$ should come closer to satisfying (3.2) than did the first set of $x_j$.

With this assumption, we may now state the iterative process:

a) Using the values at hand for $\pi_1, \pi_2, \ldots, \pi_M$, evaluate (3.5).

b) Using the values for $x_j$ obtained in step a, evaluate (5.2). If the $|g_i|$ are sufficiently small, we are done.

c) Compute $r_{i\ell}$ using (3.8) and solve (5.5) for $\Delta\lambda_i$.

d) Denoting the $\pi_i$ in step a by $\lambda_i$, we get new $\pi_i$ by (5.1).

Steps a-d are repeated until the $|g_i|$, computed in step b, are sufficiently small, or until they show no more improvement.

There is no proof of convergence for this method. In fact, the method presented here is unlikely to converge unless the starting values of $\pi_1, \pi_2, \ldots, \pi_M$ are very good, and even then there may be no convergence. This method may be used on the chemical equilibrium problem after the first-order method has resulted in a reasonably good solution. If the $\pi_i$ obtained from (3.10) in the final iteration of the first-order method are used to initiate the second-order method, the accuracy produced by the second-order method will generally be better than that which could be achieved by use of the first-order method only.

## 6. THE SECOND-ORDER CHEMICAL EQUILIBRIUM ALGORITHM

In order that the second-order linear-logarithmic method be set in the form of a chemical equilibrium problem, the same definitions as given in Sec. 4--i.e., Eqs. (4.1) through (4.5)--are used here. Since the second-order method is best used after the first-order method has been applied, the initial values of $\pi_i$ for the second-order method must be specified. The first-order method gives a set of $\pi_i'$ which are related to $\pi_i$ by Eq. (4.6). The $\pi_i$ computed by means of (4.6) are appropriate initial values for the second-order method. Using these initial values for $\pi_i$, the second-order chemical equilibrium algorithm is an iterative process for which each iteration consists of the following steps:

1) Using the current values for $(\pi_1, \pi_2, \ldots, \pi_M)$, evaluate $x_1, x_2, \ldots, x_n$ by means of (3.5).

2) Calculate $g_1, g_2, \ldots, g_m$ by means of (5.2) and set $g_{m+1}, g_{m+2}, \ldots, g_M$ equal to zero.

3) Compute $r_{i\ell}$ from (4.8) and solve (5.5) for $\Delta\lambda_1, \Delta\lambda_2, \ldots, \Delta\lambda_M$.

4) Let

$$P = \max_{i=1}^{M} |\Delta\lambda_i| \ .$$

If $P < \delta$, where $\delta$ is a small positive number such as $10^{-5}$, we are done; otherwise, let $Q = \min\left(\frac{1}{P}, 1\right)$.

5) Replace $\pi_i$ by $\pi_i + Q\ \Delta\lambda_i$ for $i = 1,2,\ldots,M$.

Steps 1-5 are repeated until the test at 4) is satisfied. P should decrease at every iteration; however, when the values for $\pi_i$ get close to their optimal values, P may not become zero due to round-off error. In order to prevent an endless repetition of steps 1-5 due to the selection of too small a $\delta$, we can test P against the value of P at the previous iteration. If this value has increased over the previous iteration, it can be assumed that this method has obtained as accurate a solution as possible, and we can terminate the iteration process. The reason for inserting the factor Q above is to prevent the $\pi_i$ from varying too much on one iteration.

## 7. SUMMARY OF THE COMPUTATION PROCEDURE

The best method for starting the solution of the chemical equilibrium problem depends on whether an estimate for the solution vector is available. The projection method should be used when the problem being solved is a slight variation from a problem previously solved, and in this case, the values used for $y_j$ in (2.9 - 2.12) should be the solution vector to the previous problem. Even when the estimate is no better than an intuitive guess, the projection method may still be used. The linear programming method, then, may be used as a back-up if the projection method produces a non-positive component. Of course, if no estimate is available, the linear programming method would be used immediately to provide an estimate.

The recommended procedure is, then, to use the first-order method until either no further progress can be made with this method or until the amount of change becomes small from iteration to iteration, and then to use the second-order method. It has been found that, for reasonably large problems (say $m = 30$, $n = 100$), the point at which progress ceases in the first-order method usually occurs when the indicated corrections to the components of the solution vector average about one per cent of the components; that is, when (3.5) is accurate to about two significant digits. A switch to the second-order method at this point usually yields quite accurate results in two iterations of the second-order method. The second-order method usually satisfies (1.1) to an accuracy of about five significant digits on a machine that carries eight significant digits. This accuracy is typically about three orders of magnitude above what is usually obtained in experimental data.

To summarize, the typical procedure for solving a chemical equilibrium problem is the following:

1) If an estimate is available, use the projection method to obtain a feasible estimate.

2) If step 1 yields a strictly positive estimate, go to step 3, but if the projection method yields non-positive components, or if there was no initial estimate, then use the linear programming method to get an estimate.

3) Use the first-order method until one of the tests described in Section 4 is satisfied.

4) Use the second-order method as described in Section 6.

# Appendix A

## A FORTRAN-IV PROGRAM FOR SOLVING THE CHEMICAL EQUILIBRIUM PROBLEM

### GENERAL DESCRIPTION

The program described here is a set of FORTRAN-IV subroutines for solving chemical equilibrium problems. The calling sequence used is merely the statement:

```
CALL SOLVE
```

Communication of data into and out of the subroutines is accomplished by a block common statement:

```
 COMMON/SLVE/IV(30),TOL(20),NR(55,2),B(55),KN(120),X(121),C(121),
1  KL(26),NAM(25,2),A(55,121),PIE(65),V1(65),V2(65),V3(65),
2  V4(65),XMF(120),X1(121),X2(121),X3(121),XBAR(25),R(65,65)
```

The data that must be input before CALL SOLVE is executed consist of the following:

| COMMON Location | Quantity |
|---|---|
| IV(1) | m |
| IV(2) | M ( = m+p) |
| IV(3) | p |
| IV(4) | n |
| IV(6) | Number of the output unit. |

| COMMON Location | Quantity |
|---|---|
| IV(7) | Print flag: -1 = minimal amount of messages; 0 = one message per iteration step; +1 = all messages. |
| IV(9) | Maximum number of iterations to be allowed. |
| B(i) | $b_i$, i = 1,2,...,m. |
| X(j) | $y_j$, j=1,2,...,m, where $y_j$ is the initial estimate of the solution. If no estimate is available, set X(J) = 0. |
| C(j) | $c_j$, j=1,2,...,n. |
| A(i,j) | $a_{ij}$, i=1,2,...,m; j=1,2,...,n. |

In addition, all components in one compartment must have consecutive subscripts. That is, components $1,2,3,\ldots,k_1$ must be in compartment 1; components $k_1+1$, $k_1+2$, ..., $k_2$ must be in compartment 2; ...; and components $k_{p-1}+1$, $k_{p-1}+2$, ..., $k_p$ must be in compartment p. These k's are communicated to the subroutines by setting

$$
\begin{aligned}
KL(1) &= 1 \\
KL(2) &= k_1+1 \\
KL(3) &= k_2+1 \\
&\vdots \\
KL(p) &= k_{p-1}+1 \\
KL(p+1) &= k_p+1 \; .
\end{aligned}
$$

In other words, KL(k) is the number of the first component in compartment k, and KL(p+1) is equal to n+1.

The above are the only numbers that need be set in order that CALL SOLVE will solve the chemical equilibrium problem. However, in order that the program can write messages, in cases of infeasibility, etc., names for the rows, components, and compartments may be input:

| COMMON Location | Quantity |
|---|---|
| NR(I,1), NR(I,2) | Two-word row name for row I. |
| KN(J) | One-word component name for component J. |
| NAM (K,1), NAM(K,2) | Two-word compartment name for compartment K. |

In addition, TOL(1) through TOL(5) are tolerances used by the program. If they are zero when the program is entered, they are set by the subroutines to nominal values. These values may also be set by the user of the subroutines, in which case the nominal values will not be set in the subroutines. These tolerances are the following:

| Tolerance | Nominal Value | Meaning |
|---|---|---|
| TOL(1) | 0.01 | $\epsilon$ in step 3 of the first-order method (see Sec. 4). |

| Tolerance | Nominal Value | Meaning |
|---|---|---|
| TOL(2) | $10^{-5}$ | δ in step 4 of the second-order method (see Sec. 6). |
| TOL(3) | $10^{-12}$ | Minimum value any $x_j$ is allowed to have. |
| TOL(4) | $10^{-6}$ | Minimum starting value that any component will have is the lesser of TOL(4) and $\frac{1}{2}y_{n+1}$ (see Sec. 2). |
| TOL(5) | $10^{-8}$ | Problem is assumed to be degenerate if any $S_k$ becomes less than TOL(5). |

With the above as input, the statement CALL SOLVE will cause an attempt to solve the chemical equilibrium problem. If, upon completion of this attempt, a solution is obtained, the cell

IV(10)

will contain a 1 and the following data will be in storage:

| COMMON Location | Data |
|---|---|
| X(i) | $x_i$, i=1,2,...,n (the solution). |
| XBAR(k) | $S_k$, k=1,2,...,p. |
| PIE(i) | $\pi_i$, i=1,2,...,m. |
| XMF(i) | $\hat{x}_i$, i=1,2,...,n. |

If IV(10) is not 1, the subroutines have failed to solve the chemical equilibrium problem. The reason for this failure is written on output unit IV(6). In such a case, X(i) will contain the latest value of these quantities.

## SUBROUTINES

There are nine subroutines in the set used for the solution of the chemical equilibrium problem. A brief description of these subroutines follows.

### 1. Subroutine SOLVE

SOLVE is the master subroutine, and is divided into four functional segments. Each segment calls other subroutines which do specific tasks. The four segments are:

a) The projection and linear programming routines for obtaining the initial solution (lines 18-42).

b) The first-order method (lines 43-122).

c) The second-order method (lines 123-163).

d) Output messages (lines 164-203).

### 2. Subroutine BAR

BAR calculates the $S_k$.

3. Subroutine BERROR

BERROR calculates

$$g_i = b_i - \sum_{j=1}^{N} a_{ij}x_j \quad . \qquad i=1,2,\ldots,M$$

4. Subroutine DEL

DEL sets

$$w_j = \sum_{i=1}^{m} a_{ij}q_i \quad . \qquad j=1,2,\ldots,n$$

5. Subroutine RCALC

RCALC calculates the $r_{i\ell}$ array (4.8).

6. Subroutine CLOG

CLOG computes

$$\alpha_j = c_j + \log \hat{x}_j \quad . \qquad j=1,2,\ldots,n$$

7. Subroutine LP

LP sets up the linear programming problems.

8. Subroutine SIMPLE

SIMPLE solves the linear programming problems. Information is communicated to this routine via a calling sequence rather than by COMMON as in subroutines 1-7. The dimension of A in SIMPLE should agree with the dimension of A in the first seven subroutines, but all other dimensions are dummy statements.

9. Subroutine MATINV

MATINV solves simultaneous equations. As in SIMPLE, no COMMON is used. The dimension of A in MATINV should agree with that of R (not A) in SOLVE. All other dimensions are singly subscripted and are irrelevant as to magnitude.

* * *

Each of the first seven subroutines has a COMMON statement which should be the same in all seven. The dimensions of the variables in this COMMON statement may be set to the values for the largest problem to be solved. With m, M, p, and n as previously defined, these dimensions must be at least:

| Symbol | Minimum Dimension |
|---|---|
| IV | 30 |
| TOL | 20 |
| NR | (m,2) |
| B | m |
| KN | n |
| X | n+1 |
| C | n+1 |
| KL | p+1 |
| NAM | (p,2) |
| A | (m,n+1) |
| PIE | M |
| V1,V2,V3,V4 | M |
| XMF | n |
| X1,X2,X3 | n+1 |
| XBAR | p |
| R | (M,M) . |

A listing of these subroutines follows. This listing does not necessarily represent an actual program. The language used was that version of FORTRAN described in [6]. The machine used for the solution of chemical equilibrium problems was the IBM-7044, which uses a floating-point number with eight bits for the exponent and 28 bits for the sign and mantissa.

## LISTING

```
      SUBROUTINE SOLVE
      COMMON/SLVE/IV(30),TOL(20),NR(55,2),B(55),KN(120),X(121),C(121),
     1  KL(26),NAM(25,2),A(55,121),PIE(65),V1(65),V2(65),V3(65),
     2  V4(65),XMF(120),X1(121),X2(121),X3(121),XBAR(25),R(65,65)
      INTEGER PF
      EQUIVALENCE  (TOL(3),XMIN),(TOL(4),XSTART),(TOL(5),BARMIN)
      EQUIVALENCE (IV(1),M),(IV(2),MEND),(IV(3),NCOMP),(IV(4),N,NTOT),
     1 (IV(5),NIT),(IV(6),NOT),(IV(7),PF),(IV(8),ITER),(IV(9),ITMAX),
     2  (IV(10),IERROR),(IV(11),LASTCP),(IV(12),KE)
      DIMENSION DX(1),ALPHA(1),TH(1),G(1)
      EQUIVALENCE (G,V1),(DX,X1),(ALPHA,X2),(TH,X3)
      IF (TOL(1).LE.0.0)   TOL(1) = 0.01
      IF (TOL(2).LE.0.0)   TOL(2) = 1.E-5
      IF (XMIN.LE.0.0) XMIN = 1.E-12
      IF (BARMIN.LE.0.0)  BARMIN = 1.E-8
      IF (ITMAX.LE.0)  ITMAX = 40
      DO 152  J = 1, NTOT
        IF (X(J).LE.0.) GO TO 5
  152 CONTINUE
C  IF X IS STRICTLY POSITIVE,  BEGIN PROJECTION
      CALL BAR( X,XBAR )
    2 CALL BERROR(ERR)
      CALL RCALC
      CALL MATINV(R,MEND,G,-1,V2,V3,V4,KE)
      IF (KE.NE.0)  GO TO 5
      CALL DEL (DX,G)
      DO 3 K = 1,NCOMP
        KTA = KL(K)
        KTB = KL(K+1)-1
        MK = M + K
        DO 4  J = KTA,KTB
          X(J) = X(J) *  ( 1. + DX(J) + G(MK) )
          IF  (X(J).LE.0.)  GO TO 5
    4   CONTINUE
    3 CONTINUE
      GO TO 7
C  LINEAR PROGRAMMING ROUTINE
    5 CALL LP(KF;
      IF (KF.NE.0) GO TO 10006
    7 CALL BAR(X,XBAR)
      CALL CLOG(X,XBAR)
      FE2 = 1.E+20
C  FIRST ORDER METHOD LOOP
      DO 899 ITER=1,ITMAX
        CALL BERROR(ERR)
        DO 7110 I=1,MEND
          PIE(I) = 0.
 7110   CONTINUE
        DO 7111  K = 1, NCOMP
          KTA = KL(K)
          KTB = KL(K+1) - 1
          MK = M + K
          DO 7112 J = KTA, KTB
            AX = ALPHA(J) * X(J)
            PIE(MK) = PIE(MK)  + AX
            DO 7113 I = 1,M
              PIE(I) = PIE(I) + AX * A(I,J)
 7113       CONTINUE
 7112     CONTINUE
 7111   CONTINUE
```

```
      DO 7114 I = 1,M
        PIE(I) = G(I) + PIE(I)
 7114 CONTINUE
      CALL RCALC
      CALL MATINV(R,MEND,PIE,-1,V2,V3,V4,KE)
      IF(KE.NE.0) GO TO 10003
      DMAX = 1.E+20
      CALL DEL(TH,PIE)
 7105 GNORM=0.
      TDA = 0.
      FE = 0.
      DO 7104  K=1,NCOMP
        MK = M + K
        KTA = KL(K)
        KTB = KL(K+1)   -1
        DO 7103 J = KTA, KTB
          TH(J) =     TH(J) +PIE(MK)- ALPHA(J)
          GNORM = GNORM + TH(J) **2
          TH(J) = TH(J) * X(J)
          TDA = TDA + TH(J) * ALPHA(J)
          IF (X(J).LT.-DMAX*TH(J)) DMAX = -X(J)/TH(J)
          FE = FE + X(J) * ALPHA(J)
 7103   CONTINUE
 7104 CONTINUE
      EPS=   SQRT ( GNORM/FLOAT (NTOT) )
      DFE = FE - FE2
      FE2 = FE
      IF (ITER.EQ.1) GO TO 7120
      ITR = ITER - 1
      IF(PF.GE.0) WRITE(NOT,799) ITR, DFE,OPTL,EPS
 7120 OPTL =AMIN1  ( 1., .99*DMAX )
      IF(PF.GT.0)WRITE(NOT,8241)            DMAX,OPTL,TDA,ERR
      IF (EPS.LE.TOL(1))  GO TO 8269
  826 IF  (TDA.GE.0.) GO TO 8267
 8260 DO 8265  II =1,54
        DO 8301 J = 1,N
          DX(J) = AMAX1(X(J) + OPTL*TH(J)      ,XMIN)
 8301   CONTINUE
        CALL BAR(DX,XBAR)
        CALL CLOG(DX,XBAR)
        TDA = 0.
        DO 8266 J = 1,NTOT
          TDA = TDA + TH(J)*ALPHA(J)
 8266   CONTINUE
        IF(PF.GT.0)WRITE(NOT, 8262)II,OPTL,    TDA
        IF ( TDA.LT.0.)  GO TO 828
 8264   OPTL = OPTL /1.4142
 8265 CONTINUE
      CALL BAR(X,XBAR)
      GO TO 8271
  828 DO 8281 J =1,NTOT
        X(J) = DX(J)
 8281 CONTINUE
      FE = 0.
      DO 8231 J=1,N
        FE = FE + ALPHA(J)*X(J)
 8231 CONTINUE
 8288 CALL SSWTCH(5,LABEL)
      IF (LABEL.NE.2) GO TO 10004
  899 CONTINUE
```

```
C   END OF FIRST ORDER METHOD LOOP
      GO TO 10002
 6000 ITER1 = ITER + 1
      PMAX  = 1.E+20
      PMAX1 = 1.E+21
C   SECOND ORDER METHOD LOOP
      DO 6002  ITER = ITER1,ITMAX
        CALL DEL(DX,PIE)
        DO 6003  K =1,NCOMP
          MTA = KL(K)
          MTB = KL(K+1) - 1
          DO 6010 J = MTA,MTB
            XMF(J)  = EXP (  DX(J) -C(J) )
            X(J) = XMF(J)*XBAR(K)
 6010     CONTINUE
        IF  (XBAR(K).LE.BARMIN)  GO TO 10005
 6003   CONTINUE
        IF (PMAX.LE.TOL(2).OR.(PMAX.GE.PMAX1.AND.PMAX.GE.PMAX2) )
     1    GO TO 10001
        CALL BERROR(ERR)
 6006   CALL RCALC
        CALL MATINV(R,MEND,G,  -1,V2,V3,V4,KE)
        IF(KE.NE.0) GO TO 10003
        PMAX2 = PMAX1
        PMAX1 = PMAX
        PMAX = 0.
        DO 6004 I = 1,MEND
          PMAX =AMAX1 ( PMAX, ABS (G(I)) )
 6004   CONTINUE
        IF (PMAX.EQ.0.0) GO TO 10001
        ZM =AMIN1 ( 1./PMAX,1.)
        DO 6005 I =1,M
          PIE(I) = PIE(I) + ZM* G(I)
 6005   CONTINUE
        DO 6011 K = 1,NCOMP
          MK = M+K
          XBAR(K) = XBAR(K)* EXP ( ZM * G(MK) )
 6011   CONTINUE
        IF (PF.GE.0) WRITE(NOT,6099) ITER,PMAX,ERR
        CALL SSWTCH(5,LABEL)
        IF (LABEL.NE.2) GO TO 10004
 6002 CONTINUE
C  END OF SECOND ORDER METHOD LOOP
10002 IERROR = 2
      WRITE(NOT,20002)
20002 FORMAT(27H ITERAION LIMIT EXCEEDED    )
      ITER = ITMAX
      GO TO 10000
10003 IERROR = 3
      WRITE(NOT,20003) KE
20003 FORMAT(21H R MATRIX HAS NULLITY,I3)
      GO TO 10000
10004 IERROR = 4
      WRITE(NOT,20004)
20004 FORMAT(56H SOLVE ROUTINE TERMINATED BECAUSE SENSE SWITCH 5 IS DOWN
     1)
      GO TO 10000
10005 IERROR = 5
      WRITE(NOT,20005)     NAM(K,1),NAM(K,2)
20005 FORMAT(13H COMPARTMENT ,2A6,10H TOO SMALL )
```

```
      LASTCP = K
      GO TO 10000
10006 IERROR = 6
      GO TO 10000
10001 IERROR = 1
10000 RETURN
 8241 FORMAT(15H    LAMBDA MAX=1PE12.4,13H, OPT LAMBDA=E10.3,6H, TDA=E12
     1.5,16H, MAX ROW ERROR=E12.5)
 8267 IF (PF.GE.0) WRITE (NOT,8268) ITER
 8268     FORMAT(10H ITERATION,I4,30H POSITIVE TDA, GO TO METHOD 2 )
         GO TO 6000
 8269 IF (PF.GE.0) WRITE (NOT,8270) ITER
 8270 FORMAT(10H ITERATION,I4,42H AV THETA LESS THAN TOL(1), GO TO METHO
     1D 2)
      GO TO 6000
 8271 IF (PF.GE.0) WRITE (NOT,8272) ITER
 8272 FORMAT(10H ITERATION,I4,36H STEP SIZE TOO SMALL, GO TO METHOD 2)
      GO TO 6000
 8262      FORMAT(10X, 4HSTEP,I2, 9H  LAMBDA=1PE10.3,6H, TDA=E15.8)
  799   FORMAT(10H ITERATION,I4,24H  CHANGE IN FREE ENERGY=1PE15.8,12H
     1STEP SIZE=E15.8,10H AV THETA=E12.5)
 6099 FORMAT(10H ITERATION,I4,19H MAX CHANGE IN PIE=1PE15.8,15H MAX ROW
     1ERROR=E15.8
      END
      SUBROUTINE BAR(W,WBAR)                                            W0001
      COMMON/SLVE/IV(30),TOL(20),NR(55,2),B(55),KN(120),X(121),C(121),  W0002
     1  KL(26),NAM(25,2),A(55,121),PIE(65),V1(65),V2(65),V3(65),        W0003
     2  V4(65),XMF(120),X1(121),X2(121),X3(121),XBAR(25),R(65,65)       W0004
      EQUIVALENCE (IV(1),M),(IV(2),MEND),(IV(3),NCOMP),(IV(4),N,NTOT),  W0005
     1 (IV(5),NIT),(IV(6),NOT),(IV(7),PF),(IV(8),ITER),(IV(9),ITMAX),   W0006
     2  (IV(10),IERROR),(IV(11),LASTCP),(IV(12),KE)                     W0007
      DIMENSION W(1),WBAR(1)                                            W0008
    7 DO 701 K = 1,NCOMP                                                W0009
        KTA = KL(K)                                                     W0010
        KTB = KL(K+1) - 1                                               W0011
        WBAR(K) = 0.                                                    W0012
        DO 702 J = KTA,KTB                                              W0013
          WBAR(K) = WBAR(K) + W(J)                                      W0014
  702   CONTINUE                                                        W0015
  701 CONTINUE                                                          W0016
      END                                                               W0017
      SUBROUTINE BERROR(BMAX)
      COMMON/SLVE/IV(30),TOL(20),NR(55,2),B(55),KN(120),X(121),C(121),
     1  KL(26),NAM(25,2),A(55,121),PIE(65),V1(65),V2(65),V3(65),
     2  V4(65),XMF(120),X1(121),X2(121),X3(121),XBAR(25),R(65,65)
      EQUIVALENCE (IV(1),M),(IV(2),MEND),(IV(3),NCOMP),(IV(4),N,NTOT),
     1 (IV(5),NIT),(IV(6),NOT),(IV(7),PF),(IV(8),IIER),(IV(9),IIMAX),
     2  (IV(10),IERROR),(IV(11),LASICP),(IV(12),KE)
      DIMENSION G(1)
      EQUIVALENCE (G,V1)
      DO 101 I = 1,M
        ZT = 0.
        DO 102 J = 1,N
          IF(A(I,J).NE.0.) ZT = ZT - X(J) * A(I,J)
  102   CONTINUE
        G(I) = ZT + B(I)
  101 CONTINUE
      DO 110 K = 1,NCOMP
        ZT = 0.
        MTA = KL(K)
        MTB = KL(K+1) - 1
        DO 111 J = MTA,MTB
          ZT = ZT + X(J)
  111   CONTINUE
        MK = M + K
        G(MK) = XBAR(K) - ZT
  110 CONTINUE
      BMAX = 0.
      DO 120 I = 1,MEND
        IF (ABS(G(I)).GT. ABS(BMAX) ) BMAX = G(I)
  120 CONTINUE
      RETURN
      END
      SUBROUTINE DEL(W,Q)
      COMMON/SLVE/IV(30),TOL(20),NR(55,2),B(55),KN(120),X(121),C(121),
     1  KL(26),NAM(25,2),A(55,121),PIE(65),V1(65),V2(65),V3(65),
     2  V4(65),XMF(120),X1(121),X2(121),X3(121),XBAR(25),R(65,65)
      EQUIVALENCE (IV(1),M),(IV(2),MEND),(IV(3),NCOMP),(IV(4),N,NIOT),
     1 (IV(5),NIT),(IV(6),NOT),(IV(7),PF),(IV(8),ITER),(IV(9),ITMAX),
     2  (IV(10),IERROR),(IV(11),LASTCP),(IV(12),KE)
      DIMENSION W(1),Q(1)
      DO 20 J = 1,N
        WW=0.
        DO 10 I = 1,M
          IF (A(I,J).NE.0.)  WW = WW + A(I,J) * Q(I)
10      CONTINUE
        W(J) = WW
20    CONTINUE
      RETURN
      END
      SUBROUTINE RCALC
      COMMON/SLVE/IV(30),TOL(20),NR(55,2),B(55),KN(120),X(121),C(121),
     1  KL(26),NAM(25,2),A(55,121),PIE(65),V1(65),V2(65),V3(65),
     2  V4(65),XMF(120),X1(121),X2(121),X3(121),XBAR(25),R(65,65)
      EQUIVALENCE (IV(1),M),(IV(2),MEND),(IV(3),NCOMP),(IV(4),N,NIOI),
     1 (IV(5),NIT),(IV(6),NOT),(IV(7),PF),(IV(8),ITER),(IV(9),ITMAX),
     2  (IV(10),IERROR),(IV(11),LASTCP),(IV(12),KE)
COMPUTE R
      DO 1 I = 1,MEND
        DO 2 J =1,1          MEND
          R(I,J) = 0.0
    2   CONTINUE
    1 CONTINUE
      DO 10  K =1,NTOT                        a^β
        DO 11  I=1,M
          IF (A(I,K).EQ.0.) GO TO 11              MEND
          AIKX = A(I,K) * X(K)
          DO 12  J =1,1
            IF (A(J,K).NE.0.) R(I,J) = A(J,K) * AIKX + R(I,J)
   12     CONTINUE                                            a^α
   11   CONTINUE
   10 CONTINUE
      DO 20 K = 1,NCOMP
        IH = K + M
        MTA =KL(K)
        MTB =KL(K+1) - 1
        DO 21 L =MTA,MTB
          DO 22 J =1,M
            IF (A(J,L).NE.0.)  R(IH,J) = R(IH,J) + A(J,L) * X(L)
   22     CONTINUE
   21   CONTINUE                                   a^α
   20 CONTINUE
      DO 30 J = 2,MEND
        JL = J-1
        DO 31 I = 1,JL
          R(I,J) = R(J,I)
   31   CONTINUE
   30 CONTINUE
   50 RETURN
      END
```

If $r_{il} = \sum_J a_{ij}^{\beta} a_{lj}^{\alpha} y_j d_j^{-1}$

"$\alpha$" = Mass Action

"$\beta$" = Mass Balance

```
      SUBROUTINE CLOG(W,WBAR)
      COMMON/SLVE/IV(30),TOL(20),NR(55,2),B(55),KN(120),X(121),C(121),
     1  KL(26),NAM(25,2),A(55,121),PIE(65),V1(65),V2(65),V3(65),
     2  V4(65),XMF(120),X1(121),X2(121),X3(121),XBAR(25),R(65,65)
      EQUIVALENCE (IV(1),M),(IV(2),MEND),(IV(3),NCOMP),(IV(4),N,NTOT),
     1 (IV(5),NIT),(IV(6),NOT),(IV(7),PF),(IV(8),ITER),(IV(9),ITMAX),
     2  (IV(10),IERROR),(IV(11),LASTCP),(IV(12),KE)
      DIMENSION  W(1),WBAR(1),ALPHA(1)
      EQUIVALENCE (X2,ALPHA)
      DO 1 K = 1, NCOMP
        KLA = KL(K)
        KLB = KL(K+1)-1
        DO 2 J = KLA,KLB
          ALPHA(J) = C(J)
          XXX = W(J)/WBAR(K)
          IF(XXX.GT.0.0) ALPHA(J) = C(J)+ALOG(XXX)
    2   CONTINUE
    1 CONTINUE
      RETURN
      END
      SUBROUTINE LP   (MON)
      COMMON/SLVE/IV(30),TOL(20),NR(55,2),B(55),KN(120),X(121),C(121),
     1  KL(26),NAM(25,2),A(55,121),PIE(65),V1(65),V2(65),V3(65),
     2  V4(65),XMF(120),X1(121),X2(121),X3(121),XBAR(25),R(65,65)
      INTEGER PF
      EQUIVALENCE  (TOL(3),XMIN),(TOL(4),XSTART),(TOL(5),BARMIN)
      EQUIVALENCE (IV(1),M),(IV(2),MEND),(IV(3),NCOMP),(IV(4),N,NTOT),
     1 (IV(5),NIT),(IV(6),NOT),(IV(7),PF),(IV(8),ITER),(IV(9),ITMAX),
     2  (IV(10),IERROR),(IV(11),LASTCP),(IV(12),KE)
      DIMENSION    XX(1),KOUT(7),CC(1),P(1)
      EQUIVALENCE(CC,XMF),(XX,X2),(P,V1)
      MON= 0
      IF (XSTART.LE.0.0) XSTART = 1.E-6
      DO 10 I = 1,M
        P(I) = B(I)
      A(I,NTOT+1) = 0.0
        DO 15 J = 1,NTOT
          A(I,NTOT+1) = A(I,NTOT+1) + A(I,J)
   15   CONTINUE
   10 CONTINUE
      DO 1  J = 1,NTOT
        CC(J) = 0.0
    1 CONTINUE
      CC(N+1) = -1.0
C  ZERO-TH SIMPLEX IS TO DETERMINE FEASIBILITY
      CALL SIMPLE(0,M,N+1,A,P,CC,KOUT,XX,PIE,V2,V3,V4,X3,R)
      ZT = XX(N+1)
      IF(PF.GE.0)WRITE (NOT,106)KOUT(2),ZT,KOUT(1)
  106 FORMAT(12H0SIMPLEX  0,I4,29H ITERATIONS, MAX MIN ELEMENT=1PE15.8,
     1 12H, CONDITION ,I3)
      ZZT   =AMIN1(ZT/2.0, XSTART)
      DO 104 I = 1,M
        P(I) = P(I) - ZZT*A(I,N+1)
  104 CONTINUE
  200 DO 201  J = 1,NTOT
        X(J) = XX(J)
        XMF(J) = 1.0
  201 CONTINUE
      IF (ZT.LE.0..OR.KOUT(1).NE.0)  GO TO 40
C   SIMPLEX LOOP
      FR2=1.E+20
      DO 301 NN = 1, NCOMP
        DO 302 J = 1, NTOT
          CC(J)     = C(J) + XMF(J) - 1.0
  302   CONTINUE
      FN = FLOAT(NN) - 1.0
      CALL SIMPLE(1,M,N   ,A,P,CC,KOUT,XX,PIE,V2,V3,V4,X3,R)
        IF (KOUT(1).NE.0) GO TO 50
  300   DO 303 J = 1,NTOT
        X(J) = XX(J)
      X(J) = ( FN*X1(J) + X(J) ) / (FN + 1.0)
          X1(J) = X(J)
  303   CONTINUE
        CALL BAR(X,XBAR)
        K = 1
      FR = 0.0
        DO 310  J = 1,N
          IF (J.GE.KL(K+1))  K = K + 1
          IF (J.EQ.KL(K).AND.XBAR(K).GT.0.0)FR=FR-XBAR(K)*ALOG(XBAR(K))
            IF (X(J).GT.0.0)  FR = FR + X(J)*(ALOG(X(J)) + C(J) )
          XMF(J) = 0.
          IF ( XBAR(K).NE.0.) XMF(J) = X(J) / XBAR(K)
  310   CONTINUE
        IF (PF.GE.0) WRITE(NOT,305) NN,KOUT(2),FR
  305   FORMAT(8H SIMPLEX,I3,1H,,I4,12H ITERATIONS ,8H FR ENG=1PE15.8)
        IF (FR.GE.FR2) GO TO 399
        FR2=FR
  301 CONTINUE
  399 DO 400 J = 1,N
        X(J) = X(J) + ZZT
  400 CONTINUE
      RETURN
   40 IF (KOUT(1).GT.1)  GO TO 50
      WRITE (NOT,41)
   41 FORMAT(72H0THIS PROBLEM IS INFEASIBLE.  THE FOLLOWING LINEAR COMBI
     1NATION OF ROWS, /1X)
      DO 140 I =1,M
        IF (PIE(I).NE.0.)  WRITE(NOT,141) PIE(I),NR(I,1),NR(I,2)
  141   FORMAT(10X,3H+ (,F15.8,5H ) * ,2A6)
  140 CONTINUE
      WRITE (NOT,142)
  142 FORMAT(48H0  LEADS TO THE FOLLOWING INFEASIBLE EQUATION,  /1X)
      DO 150 K =1,NCOMP
        MTA = KL(K)
        MTB = KL(K+1) - 1
        DO 151 J = MTA, MTB
          D = 0.
          DO 152 I =1,M
            D = PIE(I)* A(I,J) + D
  152     CONTINUE
          IF (D.NE.0.) WRITE (NOT,143) D,KN(J),NAM(K,1),NAM(K,2)
  143 FORMAT(10X,3H+ (,F15.8,5H ) * ,A6,4H IN ,2A6)
  151   CONTINUE
  150 CONTINUE
      D = 0.
      DO 160 I =1,M
        D = PIE(I)*B(I) + D
  160 CONTINUE
      WRITE (NOT,144) D
  144 FORMAT(1H0,15X, 7H+ 0.0 =,F15.8)
   70 MON = 1
      RETURN
   50 IF (KOUT(1).NE.2)  GO TO 60
      JT = KOUT(7)
      DO 51  K = 1,NCOMP
        IF ( JT.GE.KL(K))  GO TO 52
   51 CONTINUE
   52 WRITE (NOT,952)  KN(JT),NAM(K,1),NAM(K,2)
  952 FORMAT(14H THE VARIABLE ,A6,4H IN ,2A6,33H IS UNBOUNDED AND MUST B
     1E REMOVED)
      GO TO 70
   60 WRITE (NOT,960)
  960 FORMAT(60H SIMPLEX ROUTINE HAS FAILED DUE TO EXCESSIVE ROUND-OFF E
     1RROR)
      GO TO 70
      END
```

Calling Sequence for Simplex Subroutine

The simplex subroutine, SIMPLE, may be used to solve a general linear programming problem of the form: Minimize

$$\sum_{j=1}^{n} C_j x_j \qquad (1)$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \ . \qquad i=1,2,3,\ldots,m \qquad (2)$$

The $a_{ij}$ is stored in a two-dimensional array, A, with $a_{ij}$ in cell A(i,j); $C_j$ is stored in a one-dimensional array, C, with $C_j$ in cell C(j); and $b_i$ is stored in a one-dimensional array, B, with $b_i$ in cell B(i).

The calling sequence is

```
CALL SIMPLE(II,M,N,A,B,C,KO,X,P,JH,XX,Y,PE,E)
```

where

II = 0;

M = No. of rows, m;

N = No. of variables, n;

A, B, C Are as above;

KO = A subscripted variable of dimension 7;

X = A subscripted variable of dimension n or more;

P, JH, XX, Y, and PE = Subscripted variables of dimension m or more; and

E = A subscripted variable of dimension $m^2$ or more.

Upon exiting from the subroutine,

X(1),X(2),...,X(n) Contains $x_1, x_2, \ldots, x_n$ (the solution);

P(1),P(2),...,P(m) Contains the shadow prices;

KO(1) Contains an 0 if the problem was feasible, 1 if the problem was infeasible, 2 if the problem had an infinite solution, and 3, 4, or 5 if the algorithm did not terminate;

KO(2) The number of iterations taken;

KO(3) The number of pivots performed since the last inversion;

KO(4) The number of inversions performed;

KO(5) The number of pivot steps performed;

KO(6) A logical variable that is "true" if and only if the problem was feasible; and

KO(7) Contains, if the problem had an infinite solution, the number of the variable that was infinite.

The dimension of A (line X0009) must agree (at least in the first subscript) with the dimension of A in the calling program. The other dimensions need not agree with those of the calling program.

If an initial basis is available, this basis may be communicated to the subroutine by letting

$$\mathrm{II} = 1\ ,$$

$$X(i) = \begin{cases} 0.0 \text{ if variable i is not in basis,} \\ \text{(non-zero) if variable i is in basis,} \end{cases}$$

and the other quantities remain as above.

This subroutine differs from other linear programming routines in several respects. If the restraints (2) are linearly dependent, the problem is considered to be infeasible. This is the case because the chemical equilibrium problem cannot be solved if the restraints are dependent. In addition, this subroutine was written to be as scale-free as possible; this was accomplished by computing tolerances internally in the subroutine.

```
C  AUTOMATIC SIMPLEX       REDUNDANT EQUATIONS CAUSE INFEASIBILITY
      SUBROUTINE SIMPLE(INFLAG,MX,NN,A,B,C,KOUT,KB,P,JH,X,Y,PE,E)
      DIMENSION           B(1),C(1),KOUT(7),JH(1),X(1),P(1),Y(1),
     1  KB(1),E(1),PE(1),KO(7)
      EQUIVALENCE     (K,KO(1)),(ITER,KO(2)),(INVC,KO(3)),
     2  (NUMVR,KO(4)),(NUMPV,KO(5)),(FEAS,KO(6)),(JT,KO(7))
      EGUIVALENCE (XX,LL)
C THE FOLLOWING DIMENSION SHOULD BE THE SAME HERE AS IT IS IN CALLER.
      DIMENSION A(55,121)
      LOGICAL FEAS,VER,NEG,TRIG,KQ,ABSC
C
C                             MOVE INPUTS  ...  ZERO OUTPUTS
      DO 1341  I = 1,7
        KO(I) = 0
 1341 CONTINUE
      M = MX
      N = NN
      TEXP   =   .5**16
      NCUT   =   4*M + 10
      NVER   =   M/2   +  5
      M2 = M**2
      IF  (INFLAG.NE.0)  GO TO 1400
C* 'NEW'     START  PHASE ONE WITH SINGLETON BASIS
      DO 1402  J = 1,N
        KB(J) = 0
        KQ = .FALSE.
        DO 1403  I = 1,M
          IF (A(I,J).EQ.0.0)  GO TO 1403
          IF (KQ.OR.A(I,J).LT.0.0) GO TO 1402
          KQ = .TRUE.
 1403   CONTINUE
        KB(J) = 1
 1402 CONTINUE
 1400 IF (INFLAG.GT.1 )  GO TO 1320
      DO 1401  I =1,M
        JH (I) = -1
 1401 CONTINUE
C* 'VER'     CREATE INVERSE FROM 'KB' AND 'JH'
 1320 VER = .TRUE.
 1121 INVC = 0
 1122 NUMVR   =   NUMVR  +1
      DO 1101  I = 1,M2
        E(I) = 0.0
 1101 CONTINUE
      MM=1
      DO 1113  I = 1,M
        E(MM) = 1.0
        PE(I)  =   0.0
        X(I) =   B(I)
        IF (JH(I) .NE.0) JH(I) = -1
        MM = MM + M + 1
 1113 CONTINUE
C                       FORM INVERSE
      DO 1102  JT = 1,N
        IF (KB(JT).EQ.0)  GO TO 1102
        GO TO 600
C 600   CALL JMY
C                             CHOOSE PIVOT
 1114   TY = 0.0
        DO 1104 I = 1,M
```

```
          IF (JH(I).NE.-1)  GO TO 1104
          IF ( ABS(Y(I)).LE.TY)  GO TO 1104
          IR = I
          TY = ABS(Y(I))
 1104   CONTINUE
        KB(JT) = 0
C                        TEST PIVOT
        IF (TY.LE.TPIV)  GO TO 1102
C                        PIVOT
        JH(IR) = JT
        KB(JT) = IR
        GO TO 900
C 900   CALL PIV
 1102 CONTINUE
C                  RESET ARTIFICIALS
      DO 1109  I = 1,M
        IF (JH(I).EQ.-1)  JH(I) = 0
 1109 CONTINUE
 1200 VER = .FALSE.
C                              PERFORM ONE ITERATION
C* 'XCK'    DETERMINE FEASIBILITY
      FEAS= .TRUE.
      NEG = .FALSE.
      DO 1201  I = 1,M
        IF (X(I).LT.0.0)  GO TO 1250
        IF (JH(I).EQ.0)  FEAS = .FALSE.
 1201 CONTINUE
C* 'GET'    GET APPLICABLE PRICES
      IF (.NOT.FEAS)  GO TO 501
C                        PRIMAL PRICES
      DO 503 I = 1,M
        P(I) = PE(I)
  503 CONTINUE
      ABSC = .FALSE.
      GO TO 599
C                        COMPOSITE PRICES
 1250 FEAS = .FALSE.
      NEG  = .TRUE.
  501 DO 504  J = 1, M
        P(J) = 0.
  504 CONTINUE
      ABSC = .TRUE.
      DO 505  I = 1,M
        MM = I
        IF (X(I).GE.0.0)  GO TO 507
        ABSC = .FALSE.
        DO 508 J = 1,M
          P(J) = P(J) + E(MM)
          MM = MM + M
  508   CONTINUE
        GO TO 505
  507   IF (JH(I).NE.0) GO TO 505
        IF (X(I).NE.0.) ABSC = .FALSE.
        DO 510  J = 1,M
          P(J) = P(J) - E(MM)
          MM = MM + M
  510   CONTINUE
  505 CONTINUE
C* 'MIN'    FIND MINIMUM REDUCED COST
  599 JT = 0
```

```
         BB  =  0.0
         DO 701  J =1,N
C                                 SKIP COLUMNS IN BASIS
            IF (KB(J).NE.0)   GO TO 701
            DT = 0.0
            DO 303 I = 1,M
               IF (A(I,J).NE.0.0)  DT = DT + P(I) * A(I,J)
  303       CONTINUE
            IF (FEAS)  DT = DT + C(J)
            IF  (ABSC)  DT = - ABS(DT)
            IF (DT.GE.BB)  GO TO 701
            BB = DT
            JT = J
  701 CONTINUE
C  TEST FOR NO PIVOT COLUMN
         IF (JT.LE.0)  GO TO 203
C  TEST FOR ITERATION LIMIT EXCEEDED
         IF (ITER.GE.NCUT)  GO TO 160
         ITER = ITER +1
C* 'JMY'    MULTIPLY INVERSE TIMES A(.,JT)
  600 DO 610  I= 1,M
         Y(I) = 0.0
  610 CONTINUE
      LL = 0
      COST = C(JT)
      DO 605   I= 1,M
         AIJT = A(I,JT)
         IF (AIJT.EQ.0.) GO TO 602
         COST = COST + AIJT * PE(I)
         DO  606  J = 1,M
            LL = LL + 1
            Y(J) = Y(J) + AIJT * E(LL)
  606    CONTINUE
         GO TO 605
  602    LL = LL + M
  605 CONTINUE
C         COMPUTE PIVOT TOLERANCE
      YMAX = 0.0
      DO 620  I  =  1,M
         YMAX = AMAX1( ABS(Y(I)),YMAX )
  620 CONTINUE
      TPIV  =  YMAX * TEXP
C               RETURN TO INVERSION ROUTINE, IF INVERTING
      IF (VER)  GO TO 1114
C         COST TOLERANCE CONTROL
      IF (TRIG.AND.BB.GE.-TPIV)  GO TO 203
      TRIG = .FALSE.
      IF (BB.GE.-TPIV)  TRIG = .TRUE.
C* 'ROW'     SELECT PIVOT ROW
C AMONG EQS. WITH X=0, FIND MAXIMUM Y  AMONG ARTIFICIALS, OR, IF NONE,
C  GET MAX POSITIVE Y(I) AMONG REALS.
 1000 IR = 0
      AA = 0.0
      KQ = .FALSE.
      DO  1050  I =1,M
         IF (X(I).NE.0.0.OR.Y(I).LE.TPIV)  GO TO 1050
         IF (JH(I).EQ.0)  GO TO 1044
         IF (KQ) GO TO 1050
 1045    IF (Y(I).LE.AA)  GO TO 1050
         GO TO 1047
 1044    IF (KQ) GO TO 1045
         KQ = .TRUE.
 1047    AA = Y(I)
         IR = I
 1050 CONTINUE
      IF (IR.NE.0)  GO TO 1099
 1001 AA = 1.0E+20
C                FIND MIN. PIVOT AMONG POSITIVE EQUATIONS
      DO 1010  I   = 1,M
        IF (Y(I).LE.TPIV.OR.X(I).LE.0.0.OR.Y(I)*AA.LE.X(I) ) GO TO 1010
        AA = X(I)/Y(I)
        IR = I
 1010 CONTINUE
      IF (.NOT.NEG)  GO TO 1099
C  FIND PIVOT AMONG NEGATIVE EQUATIONS, IN WHICH X/Y IS LESS THAN THE
C MINIMUM X/Y IN THE POSITIVE EQUATIONS, THAT HAS THE LARGEST ABSF(Y)
 1016 BB = - TPIV
      DO 1030  I = 1,M
        IF (X(I).GE.0..OR.Y(I).GE.BB.OR.Y(I)*AA.GT.X(I) )  GO TO 1030
        BB = Y(I)
        IR = I
 1030 CONTINUE
C  TEST FOR NO PIVOT ROW
 1099 IF   (IR.LE.0)  GO TO 207
C* 'PIV'.     PIVOT ON (IR,JT)
C                           LEAVE TRANSFORMED COLUMN IN Y(I)
  900 NUMPV  =   NUMPV  +  1
      YI = -Y(IR)
      Y(IR) = -1.0
      LL = 0
C                                 TRANSFORM INVERSE
      DO  904  J = 1,M
        L = LL + IR
        IF (E(L).NE.0.0)  GO TO 905
        LL = LL + M
        GO TO 904
  905   XY = E(L) / YI
        PE(J) = PE(J) + COST * XY
        E(L) = 0.0
        DO 906  I = 1,M
          LL  =  LL + 1
          E(LL) = E(LL)  + XY * Y(I)
  906   CONTINUE
  904 CONTINUE
C                                 TRANSFORM X
      XY  =  X(IR) / YI
      DO 908  I  = 1, M
        XNEW = X(I) + XY * Y(I)
        IF (VER.OR.XNEW.GE.0..OR.Y(I).GT.TPIV.OR.X(I).LT.0.) GO TO 907
        X(I) = 0.0
        GO TO 908
  907   X(I) = XNEW
  908 CONTINUE
C                           RESTORE Y(IR)
      Y(IR) = -YI
      X(IR) = -XY
      IF (VER) GO TO 1102
  221 IA = JH(IR)
      IF (IA.GT.0)  KB(IA) = 0
  213 KB(JT)   = IR
```

```
      JH(IR)    = JT
      IF (NUMPV.LE.M) GO TO 1200
C  TEST FOR INVERSION ON THIS ITERATION
      INVC    = INVC    +1
      IF  (INVC.EQ.NVER)   GO TO 1320
      GO  TO  1200
C*  END OF ALGORITHM, SET EXIT VALUES
C              INFINITE SOLUTION
  207 K   =  2
      GO TO 250
C          PROBLEM IS CYCLING
  160 K   =  4
      GO TO 250
C          FEASIBLE OR INFEASIBLE SOLUTION
  203 K   =  0
  250 IF (.NOT.FEAS)  K = K + 1
      DO 1399  J = 1,N
        XX   =  0.0
        KBJ  =  KB(J)
        IF (KBJ.NE.0)  XX = X(KBJ)
        KB(J) = LL
 1399 CONTINUE
C       SET 'KOUT'
 1392 DO 1393 I = 1,7
        KOUT(I) = KO(I)
 1393 CONTINUE
      RETURN
       END
```

```
C     MATRIX INVERSION WITH ACCOMPANYING SOLUTION OF LINEAR EQUATIONS
      SUBROUTINE MATINV(A,N,B,M,INA,INB,IP,ISING)
C
      DIMENSION B(1),INA(1),INB(1),IP(1)
      LOGICAL IP
      DIMENSION A(65,65)
C
C     INITIALIZATION
      DO 20 J = 1,N
        IP(J) = .FALSE.
   20 CONTINUE
C BIG LOOP ON I
      DO 575  I = 1,N
        AMAX  = 0.0
C     SEARCH FOR PIVOT ELEMENT
        DO 105 J = 1,N
          IF  (IP(J))  GO TO 105
          DO 100 K = 1,N
            IF  (IP(K) .OR. ABS(AMAX).GE.ABS(A(J,K)) )  GO TO 100
            IROW  = J
            ICOL = K
            AMAX = A(J,K)
  100     CONTINUE
  105   CONTINUE
        IF (AMAX.EQ.0.0)  GO TO 750
        IP(ICOL) = .TRUE.
C     INTERCHANGE ROWS TO PUT PIVOT ELEMENT ON DIAGONAL
        IF (IROW.EQ.ICOL) GO TO 260
        DO 200 L = 1,N
          SWAP = A(IROW,L)
          A(IROW,L) = A(ICOL,L)
          A(ICOL,L) = SWAP
  200   CONTINUE
        IF (M.EQ.0) GO TO 260
        SWAP = B(IROW)
        B(IROW)    = B(ICOL)
        B(ICOL)    = SWAP
  260   INA(I) = IROW
        INB(I) = ICOL
C     DIVIDE PIVOT ROW BY PIVOT ELEMENT
        A(ICOL,ICOL) = 1.0
        DO 350  L = 1,N
          A(ICOL,L) = A(ICOL,L) / AMAX
  350   CONTINUE
        IF (M.NE.0)  B(ICOL) = B(ICOL) / AMAX
C     COMPLETE THE PIVOT
  380   DO 550 LL = 1,N
          IF (LL.EQ.ICOL) GO TO 550
          SWAP = A(LL,ICOL)
          A(LL,ICOL) = 0.0
          DO 450  L = 1,N
            A(LL,L) = A(LL,L) - A(ICOL,L) * SWAP
  450     CONTINUE
          IF (M.NE.0)  B(LL)   = B(LL)   - B(ICOL) * SWAP
  550   CONTINUE
  575 CONTINUE
  600 IF (M.LT.0) RETURN
C     INTERCHANGE COLUMNS
      DO 710  I = 1,N
      L = N + 1 - I
      IF (INA(L).EQ.INB(L))  GO TO 710
      IROW = INA(L)
      ICOL = INB(L)
      DO 705 K = 1,N
        SWAP = A(K,IROW)
        A(K,IROW) = A(K,ICOL)
        A(K,ICOL) = SWAP
  705   CONTINUE
  710 CONTINUE
  740 RETURN
C SINGULARITY FLAG
  750 ISING = 1 + N - I
      GO TO 600
      END
```

## Appendix B

## MATRIX NOTATION AND FURTHER PROOFS

The derivations in the preceding sections would be facilitated by the use of matrix notation rather than subscripted variables. We introduce the following symbols to correspond to the subscripted variables used in Sec. 3.

| Subscripted Variable | Matrix | Size of Matrix |
|---|---|---|
| $a_{ij}$ | A | M×N |
| $b_i$ | B | M×1 |
| $y_j$ | Y | N×1 |
| $d_j$ | D | N×1 |
| $c_j$ | C | N×1 |
| $\pi_i$ | $\pi$ | M×1 |
| $r_{i\ell}$ | R | M×M |
| $x_j$ | X | N×1 |

The single-column matrices may also be thought of as vectors. We use here the convention that an operator applied to a matrix means that the operator operates on each element of the matrix. For example, log Y is the N×1 matrix consisting of

$$\begin{pmatrix} \log y_1 \\ \log y_2 \\ \cdot \\ \cdot \\ \cdot \\ \log y_N \end{pmatrix} .$$

The superscript $^\tau$ indicates the transposition of a matrix. We assume that the elementary results of matrix theory are known. For example, it is known that the inverse of an invertable symmetric matrix is symmetric. The square diagonal matrix whose diagonal is one of the vectors previously defined will be denoted by the previously defined vector in elongated type; that is,

$$\mathbb{D} = \text{diag } (D)$$

and

$$\mathbb{Y} = \text{diag } (Y) \; .$$

Equations (3.2) and (3.7) in matrix notation are

$$A_\beta X = B \tag{B.1}$$

$$X = \mathbb{Y} \, (\mathbb{D}^{-1} A_\alpha^\tau \pi - \mathbb{D}^{-1} C - \log Y) \; . \tag{B.2}$$

To see the ease of matrix notation, we may substitute (B.2) into (B.1) to get

$$A\Upsilon\mathcal{D}^{-1}A^{T}\pi = B + A\Upsilon(\mathcal{D}^{-1}C + \log Y) \ . \tag{B.3}$$

By letting

$$R = A\Upsilon\mathcal{D}^{-1}A^{T} \tag{B.4}$$

and

$$S = B + A\Upsilon(\mathcal{D}^{-1}C + \log Y) \ , \tag{B.5}$$

we see that

$$R\pi = S \tag{B.6}$$

corresponds to (3.10).

In Sec. 4, we evaluated

$$\sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j} \tag{B.7}$$

but we did not give the details of the computation. The algebra of this evaluation is very difficult unless matrix algebra is used. In matrix notation, (B.7) is $\theta^T D Y^{-1}\theta$, where $\theta = X-Y$. From (B.2) we have

$$\theta = Y(D^{-1}A_{\alpha}^{T}\pi - D^{-1}C - \log Y) - Y . \tag{B.8}$$

Hence,

$$\begin{aligned}\theta^T D Y^{-1}\theta &= (\pi^T A_{\alpha} D^{-1} - C^T D^{-1} - \log Y^T) Y D Y^{-1}\theta - Y^T D Y^{-1}\theta \\ &= \pi^T A_{\alpha}(D^{-1} Y D Y^{-1})\theta - (C^T D^{-1} + \log Y^T) D Y Y^{-1}\theta - Y^T Y^{-1} D\theta \\ &= \pi^T A_{\alpha}\theta - (C^T D^{-1} + \log Y^T) D\theta - D^T\theta . \qquad \text{(B.9)} \quad \star\end{aligned}$$

Since $A_{\beta}X = B$, $A_{\beta}\theta = A_{\beta}X - A_{\beta}Y = B - A_{\beta}Y$. Also, in the chemical equilibrium formulation,

$$D^T\theta = \sum_{j=1}^{n}\theta_j - \sum_{j=n+1}^{N}\theta_j = \sum_{k=1}^{p}\left(\sum_{j\in\langle k\rangle}\theta_j - \theta_{k+m}\right) = 0$$

and

$\star$ But we have $A_{\alpha}\theta \neq B - A_{\beta}Y$ ∴ probably not true that directional derivative stuff is valid

$$(C^T D^{-1} + \log Y^T) D \theta$$

$$= \sum_{j=1}^{n} (c_j + \log y_j)\theta_j + \sum_{j=n+1}^{N} \log y_j(-\theta_j)$$

$$= \sum_{k=1}^{p} \left( \sum_{j \in \langle k \rangle} \theta_j (c_j + \log y_j) - \theta_k \log S_k \right)$$

$$= \sum_{k=1}^{p} \left( \sum_{j \in \langle k \rangle} \theta_j (c_j + \log y_j - \log S_k) \right)$$

$$= \sum_{j=1}^{n} \theta_j (c_j + \log \hat{y}_j) \ .$$

Hence,

$$\sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j} = \sum_{i=1}^{m} \pi_i \left( b_i - \sum_{j=1}^{n} a_{ij} y_j \right) - \sum_{j=1}^{n} \theta_j (c_j + \log \hat{y}_j) \quad \text{(B.10)}$$

in the context of the chemical equilibrium problem used in Sec. 4.

Next we wish to show that

$$\sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j} \geq 0$$

as stated in (4.14). First, we prove

<u>Lemma 1</u>: Let $y_1, y_2, \ldots, y_r$ be positive numbers and let $\theta_1, \theta_2, \ldots, \theta_r$ be any real numbers. Let

$$G = \sum_{j=1}^{r} \frac{\theta_j^2}{y_j} - \frac{\left(\sum_{j=1}^{r} \theta_j\right)^2}{\sum_{j=1}^{r} y_j} .$$

Then,

i) $G \geq 0$

ii) $G = 0$ if and only if

$$\frac{\theta_1}{y_1} = \frac{\theta_2}{y_2} = \ . \ . \ . = \frac{\theta_r}{y_r} .$$

<u>Proof</u>: Let $\alpha_j = \theta_j / y_j$, $j=1,2,\ldots,r$. Then,

$$G = \sum_{j=1}^{r} \alpha_j^2 y_j - \frac{\left(\sum_{j=1}^{r} \alpha_j y_j\right)^2}{\sum_{j=1}^{r} y_j}$$

$$= \left(\sum_{j=1}^{r} y_j\right)^{-1} \left[\left(\sum_{j=1}^{r} y_j\right)\left(\sum_{j=1}^{r} \alpha_j^2 y_j\right) - \left(\sum_{j=1}^{r} \alpha_j y_j\right)^2\right]$$

$$= \left(\sum_{j=1}^{r} y_j\right)^{-1} \left[\sum_{i=1}^{r} \left(\sum_{j=1}^{r} \left(\alpha_j^2 y_i y_j - \alpha_i \alpha_j y_i y_j\right)\right)\right]$$

$$= \left(\sum_{j=1}^{r} y_j\right)^{-1} \left[\sum_{i=1}^{r} \left(\sum_{j=1}^{i} \left(\alpha_j^2 y_i y_j - 2\alpha_i \alpha_j y_i y_j + \alpha_i^2 y_i y_j\right)\right)\right]$$

$$= \left(\sum_{j=1}^{r} y_j\right)^{-1} \left(\sum_{j<i} y_i y_j (\alpha_j - \alpha_i)^2\right) \geq 0 \ ,$$

which is result i). The proof is completed by noting that $G = 0$ if and only if $\alpha_i = \alpha_j$ for all i and j; this proves ii).

Now we can prove

<u>Theorem 1</u>: In the chemical equilibrium problem

i) $\displaystyle\sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j} \geq 0$

ii) $\displaystyle\sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j} = 0$ if and only if there exist numbers $\alpha_1, \alpha_2, \ldots, \alpha_p$ such that

a) $\theta_j = \alpha_{[j]} y_j$ $\qquad j \leq n$

b) $\theta_j = \alpha_{j-n} S_{j-n}$ . $\qquad j > n$

Proof: The proof follows by noting that for i > n

$$\theta_i = \sum_{j \in \langle i-n \rangle} \theta_j \ .$$

Then,

$$\sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j} = \sum_{j=1}^{n} \frac{\theta_j^2}{y_j} - \sum_{k=1}^{p} \frac{\theta_{k+n}^2}{S_k}$$

$$= \sum_{k=1}^{p} \left( \sum_{j \in \langle k \rangle} \frac{\theta_j^2}{y_j} - \frac{\left( \sum\limits_{j \in \langle k \rangle} \theta_j \right)^2}{\sum\limits_{j \in \langle k \rangle} y_j} \right) \geq 0 \qquad .$$

by lemma 1. Furthermore, by lemma 1, if the equality holds, then for each k there is a number $\alpha_k$ such that $\theta_j = \alpha_k y_j$ if $j \in k$. This, noting that b) follows from the fact that

$$\theta_i = \sum_{j \in \langle i-n \rangle} \theta_j \quad \text{for} \quad i>n \ ,$$

completes the proof of the theorem.

Our final result is

Theorem 2: In the chemical equilibrium problem, with $(y_1, y_2, \ldots, y_n)$ feasible and $\theta_1, \theta_2, \ldots, \theta_n$ calculated as in (4.7)

$$\text{i)} \quad \sum_{j=1}^{n} \theta_j (c_j + \log \hat{y}_j) \le 0$$

$$\text{ii)} \quad \sum_{j=1}^{n} \theta_j (c_j + \log \hat{y}_j) = 0 \quad \text{if and only if}$$

$(y_1, y_2, \ldots, y_n)$ is optimal.

Proof: i) follows from Theorem 1, (B.10), and the fact that $(y_1, y_2, \ldots, y_n)$ is feasible.

To prove ii), we assume that

$$\sum_{j=1}^{n} \theta_j (c_j + \log \hat{y}_j) = 0 \ .$$

Then,

$$\sum_{j=1}^{N} \frac{\theta_j^2 d_j}{y_j} = 0 \ ,$$

and $\theta_j$ is as in ii) of Theorem 1. Combining b) of Theorem 1 and (4.12) we have

$$\theta_{k+n} = S_k \pi'_{m+k} = \alpha_k S_k$$

or

$$\alpha_k = \pi'_{m+k} \ .$$

Next, we combine a) of Theorem 1 with (4.7) to get

$$\theta_j = y_j \left[ \sum_{i=1}^{m} \pi'_i a_{ij} - c_j - \log \hat{y}_j + \pi'_{[j]+m} \right]$$

$$= y_j \alpha_{[j]} = y_j \pi'_{[j]+m}$$

or

$$\sum_{i=1}^{m} \pi'_i a_{ij} - c_j - \log \hat{y}_j = 0 \ .$$

This last result is the optimality condition for $(y_1, y_2, \ldots, y_n)$ as given by (1.4), and this demonstrates the forward implication of ii). The converse follows from the fact that optimality implies that the objective function cannot be decreased.

## REFERENCES

1. Shapiro, N. Z., and L. S. Shapley, Mass Action Laws and the Gibbs Free Energy Function, The RAND Corporation, RM-3935-1-PR, September 1964.

2. Dantzig, G. B., Linear Programming and Extensions, The RAND Corporation, R-366-PR, August 1963. Also published by Princeton University Press, Princeton, New Jersey, 1963.

3. Kaplan, Wilfred, Advanced Calculus, Addison-Wesley Press, Inc., Cambridge, Massachusetts, 1952.

4. Clasen, R. J., The Linear-Logarithmic Programming Problem, The RAND Corporation, RM-3707-PR, June 1963.

5. White, W. B., S. M. Johnson, and G. B. Dantzig, Chemical Equilibrium in Complex Mixtures, The RAND Corporation, P-1059, October 8, 1957. Also published in J. Chem. Phys., 28 (1958) 751-755.

6. International Business Machines Corporation, "IBM 7040/7044 Operating System, FORTRAN IV Language," IBM Systems Reference Library, Form C28-6329, Poughkeepsie, New York, 1963.

7. Dantzig, G. B., and J. C. DeHaven, "On the Reduction of Certain Multiplicative Chemical Equilibrium Systems to Mathematically Equivalent Additive Systems," J. Chem. Phys., 36 (1962) 2620-2627.

8. Shapiro, N. Z., A Generalized Technique for Eliminating Species in Complex Chemical Equilibrium Calculations, The RAND Corporation, RM-4205-PR, August 1964.

9. Dantzig, G. B., J. C. DeHaven, I. Cooper, S. M. Johnson, E. C. DeLand, H. E. Kanter, and C. F. Sams, "A Mathematical Model of the Human External Respiratory System," Perspectives in Biol. & Med., 4 (1961) 324-376.

10. Maloney, J. V., Jr., M.D., J. C. DeHaven, E. C. DeLand, and G. B. Bradham, M.D., Analysis of Chemical Constituents of Blood by Digital Computer, The RAND Corporation, RM-3541-PR, April 1963.

11. DeHaven, J. C., and E. C. DeLand, Reactions of Hemoglobin and Steady States in the Human Respiratory System: An Investigation Using Mathematical Models and an Electronic Computer, The RAND Corporation, RM-3212-PR, December 1962.

12. DeHaven, J. C., E. C. DeLand, N. S. Assali, and W. Manson, Physiochemical Characteristics of Placental Transfer, The RAND Corporation, P-2565, March 1962.

13. Warga, J., "A Convergent Procedure for Solving the Thermo-Chemical Equilibrium Problem," J. Soc. Indust. Appl. Math., 11 (1963) 594-606.