FINLIE: A FORTRAN PROGRAM FOR FITTING
ORDINARY DIFFERENTIAL EQUATIONS WITH
NONLINEAR PARAMETERS TO DATA

James W. Bradley

February 1981

US ARMY ARMAMENT RESEARCH AND DEVELOPMENT COMMAND
BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

Approved for public release; distribution unlimited.
Destroy this report when it is no longer needed.
Do not return it to the originator.

Secondary distribution of this report by originating or sponsoring activity is prohibited.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, Springfield, Virginia 22161.

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute endorsement of any commercial product.
**Report Title:** FINLIE: A FORTRAN Program for Fitting Ordinary Differential Equations With Nonlinear Parameters to Data

**Authors:** James W. Bradley

**Performing Organization:** U.S. Army Ballistic Research Laboratory

**Report Date:** February 1981

**Abstract:**

This paper presents and documents a FORTRAN program FINLIE for fitting a system of ordinary differential equations (or a system of algebraic or transcendental equations) to observed data. FINLIE determines those values of the possibly nonlinear system parameters and initial conditions that yield a best fit—in the least squares sense—of the solution curves to measurements of one or more of the dependent variables. The basic fitting technique is Chapman-Kirk, with the Marquardt algorithm aiding convergence. The data from more than (Continued)
20. ABSTRACT (Continued):

one experiment can be handled simultaneously to obtain one common set of parameters and a set of initial conditions for each experiment. For each computer run, the value of any parameter or initial condition can be held fixed or adjusted by FINLIE.
TABLE OF CONTENTS

LIST OF TABLES ........................................... 5

I. INTRODUCTION ........................................... 7

II. INSIDE FINLIE: WHAT FINLIE DOES FOR THE USER ....... 15
   A. Condition for a Minimum γ ............................ 16
   B. Influence Coefficients ................................ 17
   C. Influence Equations for System (1) ................. 18
   D. Influence Equations for System (2) ................. 21
   E. An Overview ........................................... 22
   F. Differential Corrections in Space $S_1$ ............ 22
   G. Differential Corrections in Space $S$ ............... 25
   H. Differential Corrections in Space $\bar{S}$ .......... 29
   I. Steepest Descent ...................................... 30
   J. Marquardt Interpolation in Space $\bar{S}$ .......... 31
   K. Convergence Criterion ................................. 36
   L. Estimated Errors ...................................... 37
   M. The Composition of FINLIE ........................... 38

III. OUTSIDE FINLIE: WHAT THE USER MUST DO FOR FINLIE .... 43
   A. ROME: The User's Subroutine for Fitting Differential Equations ........................................... 43
   B. ROMA: The User's Subroutine for Fitting Algebraic or Transcendental Equations ......................... 48
   C. Calling Subroutine DUBLIN ............................ 51
   D. Writing the Program that Calls DUBLIN ............ 59

IV. SUMMARY .................................................. 62

V. ACKNOWLEDGEMENTS ...................................... 63

BIBLIOGRAPHY .............................................. 65

LIST OF SYMBOLS .......................................... 71

APPENDIX: Print-Out of FINLIE ............................ 77

DISTRIBUTION LIST ......................................... 93
LIST OF TABLES

I. Sample Data Points for System (3) or (4) ......... 12
II. Path from $P_0$ to $P_7$ for System (3) or (4)  
    and the Data of Table I ..................... 14
III. Influence Equations for System (3) and for  
    System (4) .................................. 20
IV. Matrix Equation for System (3) or (4),  
    Given Data from Three Rounds ............. 28
V. Typical $\lambda$ Values During a Fit ............. 35
VI. Subroutine ROME for System (3) ................ 47
VII. Subroutine ROMA for System (4) ............... 50

FIGURE

1. FINLIE and the User: a Schematic ............... 39
I. INTRODUCTION

This report presents and discusses a general-purpose FORTRAN equation-fitting program called FINLIE.

Assume that the behavior of some physical system can be adequately described by a set of equations involving one independent variable \( x \) and \( N_2 \) dependent variables \( (N_2 > 1) \). FINLIE requires that these equations be reducible to one of two forms:

(a) a system of \( N_2 \) first-order ordinary differential equations of the form

\[
\frac{dy_j}{dx} = f_j(x, Y, C) \quad [j=1, 2, \ldots N_2]
\]

(1)

where \( Y \) is the vector of \( N_2 \) dependent variables:

\[
Y = (y_1, y_2, \ldots y_{N_2})
\]

and where \( C \) is a vector of \( N_3 \) linearly independent parameters \( (N_3 > 0) \):

\[
C = (c_1, c_2, \ldots c_{N_3})
\]

(b) a system of \( N_2 \) algebraic and/or transcendental equations of the form

\[
y_j = g_j(x, Y_0, C) \quad [j = 1, 2, \ldots N_2]
\]

(2)

where \( Y_0 \) is the initial condition vector:

\[
Y_0 = (y_{10}, y_{20}, \ldots y_{N_2,0})
\]

The user writes his system (1) or (2) as a FORTRAN subroutine whose name is submitted to FINLIE. FINLIE's task is to adjust the parameters and initial conditions of (1) or (2) so as to fit the solution curves to measurements taken on one or more of the dependent variables. For system (1), no knowledge of the form of the solution is necessary. Indeed, we may in general assume that system (1) possesses no closed-form solution of the form (2). Otherwise, we would fit the solution equations rather than the differential equations.

System (1) can be linear or nonlinear in the parameters; system (2) can be linear or nonlinear in the parameters and in the initial conditions. However, linear parameters and initial conditions are not much of a challenge to FINLIE. Indeed, the word FINLIE can be viewed as an acronym for "FItting NonLinear Equations"; the program was created to handle nonlinear situations. (System (1) may also be nonlinear in the more common sense of "nonlinear in the dependent variables"; for our purposes, this is irrelevant.)
As a rather elementary example of system (1), consider:

\[
\begin{align*}
\frac{dy_1}{dx} &= \frac{1}{y_2} \\
\frac{dy_2}{dx} &= -c_1 (1+c_2y_2)y_2, \quad c_1 \neq 0
\end{align*}
\]  

(3)

Here \(N_2 = N_3 = 2\). If \(x\) is interpreted as distance, \(y_1\) as time and \(y_2\) as the magnitude of a missile's velocity, then (3) is essentially the drag equation for a horizontal flight in which the drag coefficient varies linearly with Mach number.

One of the reasons we chose this particular example is that it does possess a closed-form solution:

\[
\begin{align*}
y_1 &= y_{10} - c_2 (x - x_0) + (b/c_1) (u - 1) \\
y_2 &= (bu - c_2)^{-1}
\end{align*}
\]

where

\[
u = \exp [c_1 (x - x_0)]
\]

\[
b = c_2 + (y_{20})^{-1}
\]

(4)

In "real life" we would always fit (4)--which is of the form (2)--and forget about (3). In this report, however, we will use both (3) and (4) to illustrate our remarks.

FINLIE is given measurements on the first \(N_1\) of the \(N_2\) dependent variables; that is, on \(y_1, y_2 \ldots y_{N_1}\) where \(1 \leq N_1 \leq N_2\). The \(m\)-th data point \(R_m\) thus consists of \(N_1\) measurements at the independent variable value \(x_m\):

\[
R_m = (x_m, \bar{y}_{1m}, \bar{y}_{2m}, \ldots \bar{y}_{N_1,m})
\]

where \(\bar{y}_{jm}\) denotes the measured value of \(y_j\) at \(x_m\).

Assume that the measurements have been obtained from one or more distinct experiments, each experiment having its own initial condition vector. Because our first practical application of FINLIE was to rounds fired in an enclosed range, we will call each distinct experiment a round. By "multi-round" data, then, we mean \(NR\) sets of measurements (\(NR > 1\)), all applicable to the same system of equations and hence helping to determine the single parameter vector \(C\), but each measurement set determining its own initial condition vector.

Thus there are \(NR \times N_2\) initial conditions to be determined:

\[
\text{IC} = \{ (Y_0)_1, (Y_0)_2, \ldots (Y_0)_{NR} \}
\]

8
FINLIE requires that these initial conditions refer to the same independent variable value $x_0$ for every round. However, $x_0$ need not coincide with any value $x_m$ at which measurements were taken and $x_0$ need not even fall within the interval bounded by the smallest and largest of the $x_m$ values. (Of course, the farther $x_0$ lies from that interval, the more unreliable is the extrapolation to that point.)

We assume that within each round, the $x_m$ values increase with increasing $m$. For example, if we have two rounds with four and five data points, respectively, then

\[ x_1 < x_2 < x_3 < x_4 \]

and

\[ x_5 < x_6 < x_7 < x_8 < x_9 \]

but no demands are made on the combined ordering of the nine values. A member of the first string of inequalities above can be less than, equal to or greater than some member of the second string.

For convenience we coin the word "paramic" to mean "parameter or initial condition." Of course, the initial conditions are parameters of a sort: parameters whose values can change with $x_0$ and with the round. Thus, for example, system (4) could have been written in terms of four "parameters"; say, in the form

\[ y_1 = c_3 - c_2 x + (c_4 / c_1) z \]

\[ y_2 = (c_4^2 - c_2^{-1} \]

where $z = \exp (c_1 x)$. This form conceals the fact that the values of two of the four $c_j$'s will change with the initial conditions.

By our definition, a parameter is independent of the choice of $x_0$ and applies to (and is influenced by the measurements from) all the rounds. This is the essential condition we impose on the NR rounds to be fitted simultaneously: that the same parameter vector $C$ applies to each round. The measured data for any one round may be incapable of determining $C$ adequately; the combined rounds have a much better chance.

FINLIE's task is to find the set of paramics

\[ P = \{IC, C\} \quad (5) \]

that best fits the solution curve to the multi-round measurements.
Note that $P$ consists of $NR \times N2$ initial conditions and $N3$ parameters, a total of

$$N = (NR \times N2) + N3$$  \hspace{1cm} (6)

params. By a "best fit", we mean a least squares fit. That is, FINLIE seeks a particular set $P$--call it $\hat{P}$--that minimizes $\varepsilon$, the sum of the weighted squares of the residuals of the fit:

$$\varepsilon(P) = \sum_{m=1}^{N4} \sum_{j=1}^{N1} w_{jm} \left[ y_{jm} - y_j(x_m, P) \right]^2$$  \hspace{1cm} (7)

where

- $N4$ = the total number of data points $R_m$ for all the rounds;
- $w_{jm}$ = a non-negative weighting factor associated with $y_{jm}$;
- $y_j(x_m, P) = y_j$ evaluated at $x_m$, using the current value of $P$.

Other convenient measures of the goodness of fit include:

(a) the estimated variance of the fit $= s^2 = \frac{\varepsilon(P)}{N4 - N}$

(b) the estimated standard deviation of the fit $= s$

(c) the estimated probable error of the fit $= 0.67449 s$.

Note that for a least squares fit we must have $N4 > N$; that is, there must be more data points than params. (We also assume that the number of data points in each round exceeds $N2$, the number of initial conditions for each round.)

The function $\varepsilon$ is nondimensional. Hence, if we let

$$[\ ]_d \equiv \text{dimensions of } [\ ]$$,

Eq. (7) implies that

$$[w_{jm}]_d = [y_j^{-2}]_d$$  \hspace{1cm} (8)

If the user fails to specify the values of the weights, FINLIE will set all weights to unity. This may or may not be adequate. Usually the weights are chosen so that each term in (7) is of the same order of magnitude. This can be done by making $w_{jm}$ inversely proportional to the square of the uncertainty in measurement $\bar{y}_{jm}$.
\[ w_{jm} = \frac{K}{(\sigma_{jm})^2} \]  

where \( K \) is a nondimensional, positive—but otherwise arbitrary—number. That is, in general only relative uncertainties are needed.* Suppose, for example, that there are two measured variables:

- \( y_1 \) (furlongs), for which the uncertainty in each measurement is about ten furlongs;
- \( y_2 \) (fortnights), for which each uncertainty is about 0.1 fortnight.

If we choose \( K \) equal to, say, \((\sigma_{1m})^2\) in (9), we have

\[
\begin{align*}
  w_{1m} &= \frac{100}{100} = 1 \text{ (furlong)}^{-2} \\
  w_{2m} &= \frac{100}{0.01} = 10^4 \text{ (fortnight)}^{-2}
\end{align*}
\]

Any other weights for which \( w_{2m}/w_{1m} = 10^4 \) would work as well. In fact, any weights for which the ratio is "close" to \( 10^4 \)—say, within a factor of two larger or smaller—would probably work as well. Letting FINLIE set all weights at unity, on the other hand, would not work well at all in this situation. The \( y_1 \) measurements would then have much too great an influence on the fit; their noise would drown out the \( y_2 \) measurements.

If measurements are taken on more than one dependent variable (that is, if \( N_l > 1 \)), it may happen that for some data point \( R_m \), one or more (but not all) of the measurements is missing or is clearly very wrong. There is no need to discard the entire data point; it suffices to set the weights of any missing or outlier measurements at zero.

If we are fitting the solution system (2) to the data, FINLIE computes the values \( y_j(x_m, P) \) in (7) directly from the given expressions. If we are fitting the differential equation system (1), however, then FINLIE must obtain \( y_j(x_m, P) \) by numerical integration. When we have a choice, we pick (2) over (1) to avoid this integration: 'tis a summation devoutly to be missed.

Each time FINLIE is called by the user, it performs one iteration of its search procedure. That is, the user gives FINLIE the paramic set \( P_0 \) and FINLIE returns a set \( P_1 \). \( P_1 \) is almost certainly not the desired solution, but it should be an improvement over \( P_0 \) in the sense that \( \varepsilon(P_1) < \varepsilon(P_0) \). The user then gives FINLIE the set \( P_1 \) and gets back \( P_2 \), and so on. The process stops when a specified convergence criterion is satisfied or some computational disaster arises.

*However, for an absolute interpretation of \( \varepsilon \) and any error measure based on \( \varepsilon \) \( K \) should be 1.
To illustrate some of the above generalities, we return to our sample systems (3) and (4). Suppose that from three enclosed-range firings we obtain the data points \((x_m, \bar{y}_{1m})\) listed in Table I.

Assume that the \(x_m\) values in the table are exact but that each of the sixteen \(y_1\) measurements has an associated uncertainty \(\sigma_{1m}\) (seconds).

<table>
<thead>
<tr>
<th>(m)</th>
<th>(x_m) (metres)</th>
<th>(\bar{y}_{1m}) (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>2.0000000</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>2.0100507</td>
</tr>
<tr>
<td>3</td>
<td>2.0</td>
<td>2.0202034</td>
</tr>
<tr>
<td>4</td>
<td>3.0</td>
<td>2.0304591</td>
</tr>
<tr>
<td>5</td>
<td>4.0</td>
<td>2.0408189</td>
</tr>
<tr>
<td>6</td>
<td>-3.0</td>
<td>-0.0147728</td>
</tr>
<tr>
<td>7</td>
<td>-2.0</td>
<td>-0.0098987</td>
</tr>
<tr>
<td>8</td>
<td>-1.0</td>
<td>-0.0049746</td>
</tr>
<tr>
<td>9</td>
<td>0.5</td>
<td>0.0025064</td>
</tr>
<tr>
<td>10</td>
<td>1.5</td>
<td>0.0075577</td>
</tr>
<tr>
<td>11</td>
<td>2.0</td>
<td>0.0101027</td>
</tr>
<tr>
<td>12</td>
<td>0.0</td>
<td>3.0000000</td>
</tr>
<tr>
<td>13</td>
<td>1.0</td>
<td>3.0033506</td>
</tr>
<tr>
<td>14</td>
<td>2.0</td>
<td>3.0067358</td>
</tr>
<tr>
<td>15</td>
<td>3.0</td>
<td>3.0015611</td>
</tr>
<tr>
<td>16</td>
<td>5.0</td>
<td>3.0171031</td>
</tr>
</tbody>
</table>

Here we have \(N_k = 3\) rounds (the three firings), \(N_4 = 16\) measurements and \(N = 8\) parameters. The parameters are the six initial conditions and two parameters:

\[
P = \{(y_{10}, y_{20})_{E1}, (y_{10}, y_{20})_{E2}, (y_{10}, y_{20})_{E3}, c_1, c_2\}
\]  

(10)

where we arbitrarily let \(x_0\)--the \(x\) value at which all six initial conditions apply--be zero. The values of the eight parameters are to be adjusted so as to minimize

\[
\epsilon(P) = \sum_{m=1}^{16} w_{1m} [\bar{y}_{1m} - y_1(x_m, P)]^2
\]

Whenever only one dependent variable has been measured (\(N_1 = 1\)), the user--unless he has information to the contrary--can assume that all the uncertainties \(\sigma_{1m}\) are equal. This simplifies matters by allowing the user to set \(w_{1m} = 1\) for all \(m\). Thus, for Table I, we set...
The "measured" $\bar{Y}_{1m}$ values in Table I were actually obtained by rounding to seven decimal places the values computed from the solution system (4), using $x_0 = 0$ and

$$\hat{p} = \{(2,100)_{E1}, (0,200)_{E2}, (3,300)_{E3}, 0.01, 0.0001\}$$

The $\bar{Y}_{1m}$ values in the table are thus equal to $y_1(x_m, \hat{p})$ to the number of decimal places shown. FINLIE's task--given system (3) or (4) and the Table I data--would be to find $\hat{p}$.

FINLIE must be given another bit of information before it can begin its search for $\hat{p}$: a starting point $P_0$. For systems (3) and (4) and the Table I data, we gave FINLIE the relatively poor first estimate

$$P_0 = \{(1.5,50)_{E1}, [-0.5,250]_{E2}, (2.5,250)_{E3}, 0.02, 0\}$$

FINLIE then proceeded from $P_0$ to $P_1$ to $P_2$ and so on to $P_7$, an acceptable approximation to $\hat{p}$ (see Table II). Within the idiosyncracies of machine computation, this path from $P_0$ to $P_7$ is the same whether we fit system (3) or system (4). As one might expect in a convergent situation, the last two points ($P_6$ and $P_7$) are practically coincident. The slight discrepancy between $P_7$ and $\hat{p}$ is due almost entirely to the round-off error in the $\bar{Y}_{1m}$ data of Table I.

Unfortunately, a poor choice of $P_0$ can sometimes prevent FINLIE's ever finding $\hat{p}$. Hence a reasonable amount of labor expended in determining $P_0$ may pay dividends. For frequently recurring applications, it may be worthwhile for the user to write his own FORTRAN subroutine for extracting a first estimate $P_0$ from the data points. Usually only a few of the parametric estimates are critical for obtaining convergence to $\hat{p}$; the remaining parameters can have surprisingly poor first estimates with impunity. And for some systems of equations, the choice of $P_0$ is very nearly immaterial: all roads lead to $\hat{p}$.

A useful feature of FINLIE is its ability--at the user's request--to hold fixed the input values of any specified parameters, rather than allow those input values to be adjusted by the fitting process. Thus, for example, the effect of a given parameter--say, $c_2$ in system (3) or (4)--can be suppressed during a computer run by giving that parameter an initial value of zero and specifying that this
Table II. Path from \( P_0 \) to \( P_7 \) for System (3) or (4) and the Data of Table I.

<table>
<thead>
<tr>
<th>( y_{10}(s) )</th>
<th>( y_{20}(m/s) )</th>
<th>( y_{10}(s) )</th>
<th>( y_{20}(m/s) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_0 )</td>
<td>1.5</td>
<td>50</td>
<td>(-0.5)</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>1.998</td>
<td>69.75</td>
<td>(-0.00555497)</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>1.9999830</td>
<td>90.83</td>
<td>(-0.00005638)</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>1.9999996</td>
<td>99.18</td>
<td>(-0.00000546)</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>2.0000001</td>
<td>99.993</td>
<td>(-0.00000391)</td>
</tr>
<tr>
<td>( P_5 )</td>
<td>2</td>
<td>99.999727</td>
<td>(-0.00000010)</td>
</tr>
<tr>
<td>( P_6 )</td>
<td>2</td>
<td>99.999727</td>
<td>(-0.00000012)</td>
</tr>
<tr>
<td>( P_7 )</td>
<td>2</td>
<td>99.999727</td>
<td>(-0.00000012)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( y_{10}(s) )</th>
<th>( y_{20}(m/s) )</th>
<th>( c_1(1/m) )</th>
<th>( c_2(s/m) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_0 )</td>
<td>2.5</td>
<td>250</td>
<td>(-0.02)</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>2.9987</td>
<td>270.35</td>
<td>(-0.0338)</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>2.9999974</td>
<td>298.31</td>
<td>(-0.0059)</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>3.0000003</td>
<td>300.13</td>
<td>(-0.0082)</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>3</td>
<td>299.999242</td>
<td>(-0.009998427)</td>
</tr>
<tr>
<td>( P_5 )</td>
<td>3</td>
<td>299.999324</td>
<td>(-0.009998405)</td>
</tr>
<tr>
<td>( P_6 )</td>
<td>3</td>
<td>299.999324</td>
<td>(-0.009998405)</td>
</tr>
</tbody>
</table>

\[
10^6 e(P_n)_{n} \quad \epsilon(P_n)/\epsilon(P_{n-1}) \quad \text{std. dev.}
\]

| \( P_0 \)       | 3888205.        | \(0.00005\)     | \(0.6972\)      |
| \( P_1 \)       | 198.66          | \(0.07688\)     | \(0.0050\)      |
| \( P_2 \)       | 15.27           | \(0.21791\)     | \(0.0114\)      |
| \( P_3 \)       | 3.33            | \(0.05936\)     | \(0.00645\)     |
| \( P_4 \)       | \(0.0000013\)   | \(0.0001\)      | \(0.00157\)     |
| \( P_5 \)       | \(0.0000000023\)| \(0.00185\)     | \(0.00000398\)  |
| \( P_6 \)       | \(0.0000000023\)| \(0.9999995\)   | \(0.00000017\)  |
| \( P_7 \)       | \(0.0000000023\)| \(0.9999995\)   | \(0.00000017\)  |
value is to be retained. Since it is the user's task to program his particular version of equation set (1) or (2), we see that the above feature can save the user from programming many versions of the same equations, the versions differing only in the nature of the parameters involved. If the version programmed contains every parameter a reasonable (or only slightly unreasonable) person might ever want to consider, the programmer need never alter his program; he can always suppress unwanted parameters at will.

Of course, the user can also fix any parameter at a nonzero value. Consider, for example, the situation where some of the input parameter estimates are known to be respectable, ball-park values, while the remaining estimates are little more than wild guesses. There is no provision in FINLIE for weighting the parameter estimates. Thus when the data are especially noisy, FINLIE—in its single-minded effort to decrease ε—might very well downgrade an excellent estimate. One way to avoid (or at least to try to avoid) this difficulty is to make two computer runs. On the first run, all highly regarded parameter estimates are held fixed, so that the other parameters will be determined for these fixed values. The fixed and determined parameter values from this first run then serve as the estimates for a second run in which none of the parameters is held fixed.

The mechanics of informing FINLIE as to which, if any, of the parameters is to be held constant will be covered later.

In Section II, we discuss in more detail what FINLIE does for the user; in Section III, we discuss what the user must do for FINLIE.

II. INSIDE FINLIE: WHAT FINLIE DOES FOR THE USER

We rewrite the parameter set P of Eq. (5) in the form

\[ P = (P_1, P_2, \ldots, P_N) \]  \hspace{1cm} (11)

where the first \( NR \times N_2 \) elements of \( P \) are the initial conditions and the remaining \( N_3 \) elements are the parameters.

We can regard \( P \) as a point in an N-dimensional parameter space \( S \). Then \( \varepsilon(P) \), as defined by Eq. (7), is the value of the continuous scalar point function \( \varepsilon \) at point \( P \). For each point \( P \) in the parameter space \( S \), there corresponds a single value \( \varepsilon(P) \). FINLIE's task, given a starting point \( P_0 \), is to search \( S \) for a point \( P \) that yields a minimum value \( \varepsilon(P) \). (When more than one minimum exists, our choice of starting point \( P_0 \) usually determines whether or not \( \varepsilon(P) \) is the desired absolute minimum.)
The fitting process carried out by FINLIE can best be explained in terms of a single-round situation. Once the single-round procedure has been established, it will then be relatively easy to see how the process can be extended to any number of rounds.

Hence we introduce a single-round paramic set Q:

\[ Q = \{ q_1, q_2, \ldots, q_{N23} \} \]  
(12)

where

\[ N_{23} = N_2 + N_3, \]  
(13)

the number of paramics for a single round. The first \( N_2 \) elements of Q are the initial conditions and the remaining \( N_3 \) elements are the parameters. For our sample system (3) or (4), we have (for any one round)

\[ Q = (y_{10}, y_{20}, c_1, c_2), \]

Similarly, we introduce a single-round version of \( \varepsilon(P) \):

\[ \gamma(Q) = \sum_m \sum_{j=1}^{N_1} w_{jm} [\bar{y}_{jm} - y_j(x_m,Q)]^2 \]  
(14)

where the summation on \( m \) is over the measured data for the single round. (For Round E2 of Table I, for example, \( m \) would range from 6 to 11.) Note that the \( \gamma \) for a multi-round situation, Eq. (7), is the sum of the \( \gamma \)'s for the individual rounds:

\[ \varepsilon = \sum_{n=1}^{N_R} (\gamma)_E n \]  
(15)

For the moment, then--a rather long moment, lasting until Section II (G)--we will assume that FINLIE is handling a single-round situation: only one set of initial conditions is being determined.

A. Condition for a Minimum \( \gamma \)

We can regard \( Q \) as a point in an \( N_{23} \)-dimensional space \( S_1 \). A necessary (though insufficient) condition for point \( Q \) to yield a minimum value of \( \gamma \) is that the gradient of \( \gamma \) at that point be the zero vector:

\[ \text{grad} \, \gamma(Q) = \left( \frac{\partial \gamma(Q)}{\partial q_1}, \frac{\partial \gamma(Q)}{\partial q_2}, \ldots, \frac{\partial \gamma(Q)}{\partial q_{N23}} \right)_{S_1} = \hat{0} \]  
(16)

Thus FINLIE must seek a point that satisfies all \( N_{23} \) components of (16) simultaneously. From Eq. (14), we see that at any point \( Q \)
\[ \frac{\partial \gamma(Q)}{\partial q_k} = -2 \beta_k(Q) \quad [k=1,2,...N23] \] (17)

where

\[ \beta_k(Q) = \sum_m \sum_{j=1}^{N1} w_{jm} [\gamma_j - \gamma_j(x_m,Q)] \cdot D_{jk}(x_m,Q) \] (18)

\[ D_{jk}(x_m,Q) = \frac{\partial \gamma_j(x_m,Q)}{\partial q_k} \] (19)

and where, in our dimensional notation,

\[ [\beta_k]_d = [q_k^{-1}]_d \] (20)

\[ [D_{jk}]_d = [q_j q_k^{-1}]_d \] (21)

Thus condition (16) can be written in the form

\[ \beta_k(Q) = 0 \quad [k=1,2,...N23] \] (22)

The N23 components \( \beta_k \) define a vector:

\[ \vec{\beta}(Q) = (\beta_1(Q), \beta_2(Q), \ldots, \beta_{N23}(Q))_{S_1} \] (23)

which, from (16-17), has the direction of the negative gradient of \( \gamma \) at point \( Q \); that is, the direction in which the rate of decrease of \( \gamma \) is greatest:

\[ \vec{\beta} = -(1/2) \text{ grad } \gamma. \] (24)

\( \vec{\beta} \) is a vector point function of \( Q \). For each point \( Q \) in the paramic space \( S_1 \), there corresponds a unique vector \( \vec{\beta} \). Thus FINLIE's search for a point \( \hat{Q} \) that yields a minimum value \( \gamma(\hat{Q}) \) has become a search for a point \( \hat{Q} \) at which \( \vec{\beta} \) is zero.

B. Influence Coefficients

The partial derivatives \( D_{jk} \) in (18) are sometimes called "influence" (or "sensitivity") coefficients because they reflect the influence of the paramics on the solution curves.

To satisfy (22), FINLIE must be able to evaluate the influence coefficients at any point \( Q \) for each independent variable value \( x_m \).
The manner in which FINLIE evaluates $D_{jk}(x_0, Q)$ depends on which equation set, (1) or (2), we are fitting to the data.

C. Influence Equations for System (1)

If we give FINLIE the differential equation system (1), then we must also give FINLIE a system of differential equations for the influence coefficients. Taking the partial derivative of each side of (1) with respect to paramic $q_k$, we have

$$\frac{\partial}{\partial q_k} \left( \frac{d y_j}{dx} \right) = \frac{\partial f_j}{\partial q_k}$$

or, assuming that the order of differentiation can be reversed,

$$\frac{d D_{jk}}{dx} = \frac{\partial f_j}{\partial q_k}$$

(25)

The system (25) is subject to the initial conditions:

$$D_{jk}(x_0, Q) = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

(26)

(These initial conditions merely reflect the fact that the influence coefficient $D_{jj}$ is, by our definition, $\partial y_j/\partial y_j$ and hence equals one at $x_0$.)

The paramics affect $f_j(x, Y, C)$ in two ways: indirectly through their effect on the dependent variable vector $Y$ and directly through the parameter vector $C$. Hence (25) can be rewritten in the more cumbersome but (possibly) more revealing form:

$$\frac{d D_{jk}}{dx} = \sum_{i=1}^{N2} \left( \frac{\partial f_j}{\partial y_i} \right)_C D_{ik} + \begin{cases} 0 & \text{if } k \leq N2 \\ \left( \frac{\partial f_j}{\partial y_{j_0}} \frac{\partial y_{j_0}}{\partial c_{k-N2}} \right) & \text{if } k > N2 \end{cases}$$

(27)
where

subscript $C$ indicates that $x$ and vector $C$ are considered constant in taking the partial derivatives of $f_j(x,Y,C)$;

subscript $Y$ indicates that $x$ and vector $Y$ are considered constant in taking the partial derivatives of $f_j(x,Y,C)$.

Thus, by "paramic differentiation" we obtain an auxiliary system of differential equations (27) whose solutions are the influence coefficients needed to fit equation set (1). Note from (27) that these influence equations are always linear in the influence coefficients $D_{jk}$. The number of influence equations is

$$NA = N_2 \times N_3$$

(28)

The user must include his version of system (27) in the FORTRAN subroutine containing his version of system (1).

For our by-now-familiar example, system (3), we have $NA = 2 \times 4 = 8$. The eight influence equations for system (3) are shown in the upper portion of Table III, where $(\cdot)' = d(\cdot)/dx$.

Recall that our only purpose in obtaining the influence coefficients is to be able to evaluate $\beta_k(Q)$, Eq. (18), in our effort to satisfy condition (22). From (18) we see that $\beta_k$ involves $D_{jk}$ only for $j=1$ to $N_1$; that is, only for the measured variables. Yet Eqs. (25-27) show $j$ running from 1 to $N_2$; that is, over all the dependent variables. Do we have more influence equations here than we need? The answer is: no. We have implicitly assumed that there are no extraneous dependent variables in system (1): all of the unmeasured dependent variables are needed to solve the differential equations for the measured variables. Hence the $D_{jk}$ for $N_1 < j < N_2$ are also needed.

For our example, system (3) with $N_1=1$, $y_2$ is clearly needed to solve the differential equation for $y_1$. Thus each $D_{2k}$ is also needed, as we see in Table III (A). (On the other hand, if $y_2$ had been the only measured variable in system (3), then $y_1$ would be an extraneous variable and should be thrown out.)

The mechanics of writing and submitting the influence equations will be discussed later. FINLIE will automatically assign the proper initial conditions (26) and integrate the influence equations simultaneously with the original system (1) to obtain $y_j(x_m,Q)$ and $D_{jk}(x_m,Q)$ at each $x_m$. 

19
Table III. Influence Equations for System (3) and for System (4)

(A) For System (3):

\[
(D_{11})' = \frac{\partial y_1'}{\partial y_{10}} = -(y_2^2)D_{21}
\]

\[
(D_{12})' = \frac{\partial y_1'}{\partial y_{20}} = -(y_2^2)D_{22}
\]

\[
(D_{13})' = \frac{\partial y_1'}{\partial c_1} = -(y_2^2)D_{23}
\]

\[
(D_{14})' = \frac{\partial y_1'}{\partial c_2} = -(y_2^2)D_{24}
\]

\[
(D_{21})' = \frac{\partial y_2'}{\partial y_{10}} = -c_1(1+2c_2y_2)D_{21}
\]

\[
(D_{22})' = \frac{\partial y_2'}{\partial y_{20}} = -c_1(1+2c_2y_2)D_{22}
\]

\[
(D_{23})' = \frac{\partial y_2'}{\partial c_1} = -c_1(1+2c_2y_2)D_{23} - (1+c_2y_2)y_2
\]

\[
(D_{24})' = \frac{\partial y_2'}{\partial c_2} = -c_1(1+2c_2y_2)D_{24} - c_1y_2^2
\]

where \((D_{11})_0 = (D_{22})_0 = 1; (D_{jk})_0 = 0\) for \(j \neq k\)

(B) For System (4):

\[
D_{11} = \frac{\partial y_1}{\partial y_{10}} = 1
\]

\[
D_{12} = \frac{\partial y_1}{\partial y_{20}} = -(u-1)/(c_1y_{20}^2)
\]

\[
D_{13} = \frac{\partial y_1}{\partial c_1} = (b/c_1^2)[1-u+c_1(x-x_0)u]
\]

\[
D_{14} = \frac{\partial y_1}{\partial c_2} = (u-1)/c_1 - (x-x_0)
\]

(C) Unneeded Influence Equations for System (4):

\[
D_{21} = \frac{\partial y_2}{\partial y_{10}} = 0
\]

\[
D_{22} = \frac{\partial y_2}{\partial y_{20}} = (y_2^2/y_{20})^2 u
\]

\[
D_{23} = \frac{\partial y_2}{\partial c_1} = -by_2^2(x-x_0)u
\]

\[
D_{24} = \frac{\partial y_2}{\partial c_2} = -(u-1)y_2^2
\]
One final remark. For large systems with many paramics, the exact influence equations (27) can be rather cumbersome. In many cases, certain liberties can be taken with the influence equations: expressions can be approximated by simpler ones, the effect of certain paramics on certain terms in the original equations can be ignored, etc. If done with care and judgment, such simplifications will have no effect on the final answer: the same point Q will be reached with or without the simplifications. Note, however, that discretion is called for. If the user has any doubts as to the merits of some modification to the exact influence equations (and even when he hasn't any doubts), his safest course is to avoid such a modification.

D. Influence Equations for System (2)

If we give FINLIE the solution set (2), then we must also give FINLIE the influence equations obtained by differentiating (2):

\[ D_{jk} = \frac{d g_j}{d q_k} \quad \begin{bmatrix} j = 1, 2, \ldots, N1 \\ k = k', 2, \ldots, N23 \end{bmatrix} \]  

(29)

We assume—as with system (1)—that there are no extraneous variables in system (2). (For (2), this means that the initial conditions for all of the unmeasured dependent variables are needed to evaluate the expressions for the measured variables.) However, the \( D_{jk} \) for \( N1 < j \leq N2 \) are superfluous and should be ignored. Thus the number of influence equations required to fit system (2) is

\[ NB = N1 \times N23 \]  

(30)

To fit system (4), for example, (where \( N1 = 1 \) and \( N2 = 2 \)), the \( D_{2k} \) values are not required and we need submit only four influence equations to FINLIE. These equations are shown in Table III (B). FINLIE will automatically set all undefined \( D_{j'k} \)'s to zero. For the sake of completeness, expressions for the unneeded \( D_{jk} \) are given in Table III (C), but we emphasize that these latter equations should not be given to FINLIE. Note that the eight expressions for \( D_{jk} \) in Table III (B and C) do indeed satisfy the initial conditions indicated in part A of the table.

The remarks in the previous section on the possibility of simplifying the influence equations apply to system (29), although here the urge to simplify may be less compelling.
E. An Overview

To summarize thus far: FINLIE determines the values of $y_j(x_m, Q)$ and $D_{jk}(x_m, Q)$ either

(i) by numerically integrating a system of $N^2$ plus $NA$ first-order differential equations or

(ii) by evaluating a system of $N^2$ plus $NB$ algebraic or transcendental expressions.

Except for this difference—but what a difference it can be in terms of machine time!—the fitting process used by FINLIE is the same for the two equation sets (1) and (2).

This fitting process consists of adjusting $Q$ until it satisfies condition (22). Of course, it would be pleasant if FINLIE could solve (22) for $\hat{Q}$ in some direct, one-step fashion. No fooling around with $Q_0, Q_1, \ldots$; just leap in and solve the $N^2$ equations of (22) for the $N^2$ components of $\hat{Q}$. Unfortunately, when system (22) is nonlinear in one or more of the parameters, no such general one-step scheme exists. Hence FINLIE, expecting the worst, sets out to solve (22) by an iterative process.

Two of the standard iterative techniques are:

(i) differential corrections (alias Taylor-series linearization, alias Gauss method, alias Gauss-Newton method);

(ii) steepest descent (alias gradient search).

FINLIE uses a third method, due to Marquardt*, which is a blend of the first two methods, retaining the best features of each and avoiding their disadvantages. We will discuss enough of the differential corrections and steepest descent techniques to see what is involved in combining the two.

F. Differential Corrections in Space $S_1$

For each point $Q$ in $S_1$ there corresponds a position vector $\hat{Q}$. Let $\delta \hat{Q}$ be the vector from point $Q$ to point $\hat{Q}$:

$$\hat{Q} = \hat{Q} + \delta \hat{Q}$$

$$\hat{Q} = \hat{Q} + (\Delta q_1, \Delta q_2, \ldots, \Delta q_{N^{23}})_{S_1}$$

* See the Bibliography, Part A.
In the differential corrections technique, we approximate the basic condition (22) by a system of equations (to be derived in the next paragraph) that is linear in the increments $\Delta q_k$. We can't solve (22) for $\hat{Q}$, but given a point, say $Q_0$, we can solve the approximate conditions for an approximate increment vector $\Delta \hat{Q}_0$. This increment is then added to $\hat{Q}_0$ to reach the next way-station on our trek to $\hat{Q}$:

$$\hat{Q}_1 = \hat{Q}_0 + \Delta \hat{Q}_0 \quad (32)$$

Point $Q_1$ is an improvement over point $Q_0$ if $\gamma(Q_1) \leq \gamma(Q_0)$. But improvement or not, the differential corrections method plows ahead, solving $Q_1$ to re-solve the approximate equations for a new increment $\Delta \hat{Q}_1$. The process continues in this manner through a series of points until a specified convergence criterion has been met or a specified number of iterations have been performed or some numerical catastrophe occurs.

The desired approximation to condition (22) can be obtained by expanding $y_j$ and $D_{jk}$ in Taylor series about point $Q$. We have

$$y_j (x_m, \hat{Q}) = y_j (x_m, Q) + \sum_{n=1}^{N23} D_{jn} (x_m, Q) \cdot \Delta q_n + (\text{higher-order terms}) \quad (33)$$

$$D_{jk} (x_m, \hat{Q}) = D_{jk} (x_m, Q) + (\text{higher-order terms}) \quad (34)$$

We assume—an assumption that is not always valid—that $Q$ is close enough to $Q$ to permit us to ignore the higher-order terms in (33) and (34). Then from definition (18), we have

$$\beta_k (Q) \approx \sum_{m} \sum_{j=1}^{N1} w_{jm} \left[ y_{jm} - y_j (x_m', \hat{Q}) \right] \cdot D_{jk} (x_m', \hat{Q})$$

$$\approx \sum_{m} \sum_{j=1}^{N1} w_{jm} \left[ y_{jm} - y_j (x_m', Q) - \sum_{n=1}^{N23} D_{jn} (x_m', Q) \cdot \Delta q_n \right] \cdot D_{jk} (x_m', Q)$$

$$\approx \beta_k (Q) - \sum_{m} \sum_{j=1}^{N1} w_{jm} D_{jk} (x_m', Q) \left[ \sum_{n=1}^{N23} D_{jn} (x_m', Q) \cdot \Delta q_n \right]$$

23
By rearranging the sums, we obtain

\[ \beta_k(\hat{Q}) = \beta_k(Q) - \sum_{n=1}^{N_{23}} \alpha_{kn}(Q) \cdot \Delta q_n \]  (35)

where

\[ \alpha_{kn}(Q) = \sum_m \sum_{j=1}^{N_{11}} w_{jm} \cdot D_{jk}(x_m, Q) \cdot D_{jn}(x_m, Q) \]  (36)

Thus the conditions \( \beta_k(\hat{Q}) = 0 \), which hold at a point \( \hat{Q} \) where \( \gamma \) is at a minimum, are replaced by the conditions

\[ \beta_k(Q) = \sum_{n=1}^{N_{23}} \alpha_{kn}(Q) \cdot \Delta q_n \]  (37)

\([k = 1, 2, \ldots, N_{23}]\)

which are applicable to points in the vicinity of \( \hat{Q} \).

The quantities \( \alpha_{kn}(Q) \) have at least four interesting properties:

\[ [\alpha_{kn}]_d = [(q_k q_n)^{-1}]_d \]

\[ a_{nk} = a_{kn} \]

\[ a_{kk} > 0 \]

\[ a_{nn} a_{kk} > a_{nk}^2 \]  (38)

The first three properties follow at once from definition (36); the fourth is a consequence of Hölder's Inequality (alias Cauchy's, alias Schwarz's, alias Buniakovski's Inequality). In general, we regard \( \alpha_{kn} \) as the \((k,n)\)-th element of an \( N_{23} \) by \( N_{23} \) symmetric matrix \( \alpha \).

In matrix form, (37) becomes

\[ [\alpha(Q) \cdot \Delta \hat{Q}^T = \frac{1}{\beta}(Q)]_{S_1} \]  (39)

24
where the superscript $T$ (for Transpose) denotes a column vector and the subscript $S_1$ indicates that all components are in the $N_{23}$-dimensional space $S_1$. For either of our examples, system (3) or (4), (39) becomes

$$
\begin{pmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{21} & a_{22} & a_{23} & a_{24} \\
  a_{31} & a_{32} & a_{33} & a_{34} \\
  a_{41} & a_{42} & a_{43} & a_{44}
\end{pmatrix}
\begin{pmatrix}
  \Delta y_{10} \\
  \Delta y_{20} \\
  \Delta c_1 \\
  \Delta c_2
\end{pmatrix}
= 
\begin{pmatrix}
  \beta_1 \\
  \beta_2 \\
  \beta_3 \\
  \beta_4
\end{pmatrix}
$$

(40)

System (39) is linear in the increments $\Delta q_k$; hence the process of solving for these increments is routine work for the computer. (We assume that a solution does exist; this amounts to assuming that the determinant of matrix $a$ is not zero.)

The differential corrections process, then, consists of substituting $Q_0$ in (39), solving for $\Delta Q_0$, substituting in (39) the point $Q_1$ obtained by the vector addition $Q_1 = Q_0 + \Delta Q_0$, solving for $\Delta Q_1$, etc.

Unfortunately, even when this process converges to some point, there is no guarantee that this point will yield the absolute minimum $\gamma$. Condition (22)—which is approximated by the matrix equation (39)—guarantees only that its solution point $Q$ will yield some relative extremum value of $\gamma$. Space $S_1$ could be teeming with points of local extremum. Each of these extremum points, including the one we seek, is a sort of black hole in space $S_1$, capable of drawing a nearby search party into its core. The particular black hole into which we are drawn depends mainly on where we start in space $S_1$.

G. Differential Corrections in Space $S$

So far in Section II, we have assumed single-round data, $NR = 1$. For this situation, the differential corrections technique led to matrix equation (39).

Consider now the three-round situation of Table I. For each round $E_i$ $(i = 1, 2, 3)$, FINLIE forms a vector $b_{E_i}$ and a matrix $a_{E_i}$ by Eqs. (18) and (36) respectively, using the $Q$ and $m$ indicated below:
Range of Subscript $m$ in Eqs. (18) and (36)

<table>
<thead>
<tr>
<th>Round</th>
<th>Point $Q$</th>
<th>Range of $m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>$(y_{10}^1)<em>{E1}$, $(y</em>{20}^1)_{E1}$, $c_1$, $c_2$</td>
<td>1 to 5</td>
</tr>
<tr>
<td>E2</td>
<td>$(y_{10}^2)<em>{E2}$, $(y</em>{20}^2)_{E2}$, $c_1$, $c_2$</td>
<td>6 to 11</td>
</tr>
<tr>
<td>E3</td>
<td>$(y_{10}^3)<em>{E3}$, $(y</em>{20}^3)_{E3}$, $c_1$, $c_2$</td>
<td>12 to 16</td>
</tr>
</tbody>
</table>

In the 8-dimensional space $S$ associated with the eight parameters $p_k$ of Eq. (10), the $\hat{b}_{E_i}$ vectors take the form:

$$
\begin{align*}
\hat{b}_{E1} &= [(\beta_1^1)_{E1}, (\beta_2^1)_{E1}, 0, 0, 0, (\beta_3^1)_{E1}, (\beta_4^1)_{E1}]_S \\
\hat{b}_{E2} &= [0, 0, (\beta_1^2)_{E2}, (\beta_2^2)_{E2}, 0, 0, (\beta_3^2)_{E2}, (\beta_4^2)_{E2}]_S \\
\hat{b}_{E3} &= [0, 0, 0, (\beta_1^3)_{E3}, (\beta_2^3)_{E3}, (\beta_3^3)_{E3}, (\beta_4^3)_{E3}]_S
\end{align*}
$$

(41)

Similarly, in space $S$ the matrix $a$ for round $E1$ expands to:

$$
\begin{align*}
&\begin{pmatrix}
(a_{11}^1)_{E1} & (a_{12}^1)_{E1} & 0 & 0 & 0 & 0 & 0 & (a_{13}^1)_{E1} & (a_{14}^1)_{E1} \\
(a_{21}^1)_{E1} & (a_{22}^1)_{E1} & 0 & 0 & 0 & 0 & 0 & (a_{23}^1)_{E1} & (a_{24}^1)_{E1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(a_{31}^1)_{E1} & (a_{32}^1)_{E1} & 0 & 0 & 0 & 0 & 0 & (a_{33}^1)_{E1} & (a_{34}^1)_{E1} \\
(a_{41}^1)_{E1} & (a_{42}^1)_{E1} & 0 & 0 & 0 & 0 & 0 & (a_{43}^1)_{E1} & (a_{44}^1)_{E1}
\end{pmatrix}
\end{align*}
$$

(42)

with similar expressions for $a_{E2}$ and $a_{E3}$.

Since the multi-round $\varepsilon$ to be minimized is the sum of the single-round $\gamma$'s, FINLIE obtains the multi-round version of Eq. (40) by summing---in space $S$---the three single-round $\hat{b}_{E1}$ vectors of Eq. (41):

---

26
\[ \mathbf{B} = [\mathbf{E}_1 + \mathbf{E}_2 + \mathbf{E}_3] S \]  

and the three single-round \( a_{Ei} \) matrices:

\[ A = [a_{E1} + a_{E2} + a_{E3}] S \]  

The desired multi-round matrix equation is then

\[ [A(P) \cdot \Delta P^T = \mathbf{B}^T(P)] S \]  

A detailed form of this equation for our three-round sample system is given in Table IV; the generalization to any number of rounds can be easily visualized.

The \( N \) by \( N \) symmetric matrix \( A \) will always contain

\( N R \times (N R - 1) \times N 2 \times N 2 \)

zeroes distributed among the off-diagonal elements of all but the last \( N 3 \) rows and columns. Let \( a_{kn} \) be the \((k,n)\)-th element of matrix \( A \). As in Eqs. (38), we have

\[ [a_{kn}]_d = [(p_{kPn})^{-1}]_d \]

\[ a_{nk} = a_{kn} \]

\[ a_{kk} > 0 \]

\[ a_{nn} a_{kk} > a_{nk}^2 \]

Similarly, if \( b_k \) denotes the \( k \)-th component of vector \( \mathbf{B} \), then

\[ [b_k]_d = [p_k^{-1}]_d \]

We have taken some pains to distinguish between the multi-round paramics \( p_k \) and the single-round paramics \( q_k \), which for our three-round sample systems take the form

\[ p = [(y_{10}, y_{20})_{E1}, (y_{10}, y_{20})_{E2}, (y_{10}, y_{20})_{E3}, c_1, c_2) \]

\[ q = (y_{10}, y_{20}, c_1, c_2) \]
Table IV. Matrix Equation for System (3) or (4), Given Data from Three Rounds

\[
\begin{pmatrix}
(a_{11})_{E1} & (a_{12})_{E1} & 0 & 0 & 0 & 0 & (a_{13})_{E1} & (a_{14})_{E1} \\
(a_{21})_{E1} & (a_{22})_{E1} & 0 & 0 & 0 & 0 & (a_{23})_{E1} & (a_{24})_{E1} \\
0 & 0 & (a_{11})_{E2} & (a_{12})_{E2} & 0 & 0 & (a_{13})_{E2} & (a_{14})_{E2} \\
0 & 0 & (a_{21})_{E2} & (a_{22})_{E2} & 0 & 0 & (a_{23})_{E2} & (a_{24})_{E2} \\
0 & 0 & 0 & 0 & (a_{11})_{E3} & (a_{12})_{E3} & (a_{13})_{E3} & (a_{14})_{E3} \\
0 & 0 & 0 & 0 & (a_{21})_{E3} & (a_{22})_{E3} & (a_{23})_{E3} & (a_{24})_{E3} \\
(a_{31})_{E1} & (a_{32})_{E1} & (a_{31})_{E2} & (a_{32})_{E2} & (a_{31})_{E3} & (a_{32})_{E3} & S_{33} & S_{34} \\
(a_{41})_{E1} & (a_{42})_{E1} & (a_{41})_{E2} & (a_{42})_{E2} & (a_{41})_{E3} & (a_{42})_{E3} & S_{43} & S_{44}
\end{pmatrix} \begin{pmatrix}
(a_\gamma)_{10}^{E1} \\
(a_\gamma)_{20}^{E1} \\
(a_\gamma)_{10}^{E2} \\
(a_\gamma)_{20}^{E2} \\
(a_\gamma)_{10}^{E3} \\
(a_\gamma)_{20}^{E3} \\
\Delta c_1 \\
T_3 \\
T_4
\end{pmatrix} = \begin{pmatrix}
(b_1)_{E1} \\
b_2_{E1} \\
b_1_{E2} \\
b_2_{E2} \\
b_1_{E3} \\
b_2_{E3} \\
\Delta c_2 \\
T_4
\end{pmatrix}
\]

where \((\_)_ {E_i}\) denotes the value of \((\_)\) based solely on the measured data from round \(E_i\) \((i=1,2,3)\).

\[S_{kn} = (a_{kn})_{E1} + (a_{kn})_{E2} + (a_{kn})_{E3}\]

\[T_k = (b_k)_{E1} + (b_k)_{E2} + (b_k)_{E3}\]
The chief reason for taking these pains is that the FINLIE user must himself make this distinction in a multi-round situation. The FINLIE input arguments (to be discussed later) are defined in terms of the \( N \) paramics \( p_k \), but the influence equations submitted to FINLIE must always be written in terms of the \( N_{23} (= N_2 + N_3) \) paramics \( q_k \). The values of the initial conditions may change with the round, but the influence equations themselves, like the original equations (1) or (2) on which they are based, remain the same. Thus, regardless of the number of rounds, there will always be \( N_{23} \) influence coefficients, defined in terms of the \( N_{23} \) paramics \( q_k \), and there will always be \( NA (= N_2 \times N_{23}) \) or \( NB (= N_1 \times N_{23}) \) influence equations (depending on whether the user is working with system (1) or system (2)).

**H. Differential Corrections in Space \( \tilde{S} \)**

If the paramics \( p_k \) are not all of the same dimension, our paramic space \( S \) is a hodgepodge: a salmagundi, a gallimaufry, an olla-podrida of units. Certain computational advantages can be obtained by working in a space \( \tilde{S} \) in which the paramics—and hence the components of grad \( \varepsilon \)—are nondimensional. (The advantages of \( \tilde{S} \) are especially compelling in the steepest descent technique, some of whose properties are not scale-invariant.)

To achieve the desired paramic transformation from \( S \) to \( \tilde{S} \), we note from Eqs. (46) that

\[
[a_{kk}]_d = [p_k^{-2}]_d
\]

or

\[
[(a_{kk})^{1/2} p_k] = 1
\]

That is, the bracketed quantity in (48) is nondimensional. Thus the paramic transformation

\[
\tilde{p}_k = (a_{kk})^{1/2} p_k
\]

creates the desired* paramic space \( \tilde{S} \). The elements of \( A \) and \( \tilde{A} \) in \( \tilde{S} \) are

*From Eq. (47), we see that the product \( b_k p_k \) is also nondimensional.

Thus, the transformation 

\[
\tilde{p}_k = b_k p_k
\]

seems appealing; it would lead to a space in which all components of \( \tilde{A} \) are unity. The appeal, however, is illusory. It would not be very wise to use as scale factors the very quantities \( b_k \) that we are trying to drive to zero. The scale factors \( (a_{kk})^{1/2} \), on the other hand, are never zero (see Eq. (46)).
\[ \tilde{a}_{jk} = (a_{jj} a_{kk})^{-\frac{1}{2}} a_{jk} \]  
\[ \tilde{b}_k = (a_{kk})^{-\frac{1}{2}} b_k \]  

These space \( \tilde{S} \) components have the following admirable features:

(i) \( \tilde{p}_k, \tilde{a}_{jk} \) and \( \tilde{b}_k \) are nondimensional;
(ii) the diagonal elements of matrix \( A \) are unity:
\[ \tilde{a}_{kk} = 1 \]  
(iii) the off-diagonal elements of \( A \) satisfy the inequality:
\[ -1 \leq \tilde{a}_{jk} \leq 1 \]  

Finally, the form of matrix equation (45) is unchanged:
\[ \left[ \begin{array}{c} A(P) \cdot \Delta P^T = \hat{B}^T(P) \end{array} \right] \]  

the subscript \( \tilde{S} \) serving to remind us that all components are now in the scaled paramic space. Ninety-nine percent of the labor in solving (54) for \( \Delta P \) is usually expended in inverting matrix \( A \). Use of the scaled components \( \tilde{a}_{jk} \) tends to increase the accuracy of the matrix inversion process.

Note that each scale factor \( (a_{kk})^{\frac{1}{2}} \) in (49) is a function of point \( P \), the current set of paramic values. Hence each time the parameters are up-dated, a new transformation must be made: a new \( \tilde{S} \) space created. This is no big problem for a computer. FINLIE handles the scaling to space \( \tilde{S} \) and back again to the user's space \( S \); the process is automatic and invisible (in computer jargon, "transparent") to the user.

I. Steepest Descent

Consider a given point \( P_0 \) and the corresponding vector \( \hat{B}(P_0) \) proceeding from that point. Recall that \( \hat{B} \) at any point is a vector in the direction of the negative gradient of \( \epsilon \) at that point. Hence, provided that the magnitude of \( \hat{B}(P_0) \) is not zero (if it were, \( P_0 \) would be the desired solution \( \hat{P} \) ), \( \hat{B}(P_0) \) is the steepest descent vector for point \( P_0 \): a vector in whose direction \( \epsilon(P) \) will decrease most rapidly (at least at first) as we move away from \( P_0 \). Let \( P_1 \) be any other point
in this steepest descent direction:

\[ \hat{P}_1 = \hat{P}_0 + h \cdot \hat{b}(P_0) \]  

where \( h \) is a nondimensional positive constant.

There always exists a range of \( h \) values, \( 0 < h < h_{\text{max}} \), for which the point \( P_1 \) obtained by (55) is an improvement: \( \varepsilon(P_1) < \varepsilon(P_0) \). The steepest descent method determines the optimum \( h \) in this range: the value of \( h \) for which \( \varepsilon \) is a local minimum along the vector \( \hat{b}(P_0) \). This can be done by evaluating \( P_1 \) and \( \varepsilon(P_1) \) for a series of \( h \) values:

\[ h_0 < h_1 < h_2 < \ldots \]  

Presumably, for a while \( \varepsilon \) will decrease with increasing \( h \). As soon as an \( h \) is found for which the \( \varepsilon \) has increased, the (approximately) optimum \( h \) for point \( P_0 \) can be determined by interpolation.

Given the new point \( P_1 \) based on this optimum \( h \), the next point \( P_2 \) will lie in the direction of steepest descent from \( P_1 \); that is, along the new vector \( \hat{b}(P_1) \). Another optimum \( h \) must be determined to obtain \( P_2 \). And so on to \( P \).

The difficulty with this approach is that in the neighborhood of the solution point \( P \), where \( |\hat{b}| \) is nearly zero and yet we are not quite close enough to \( P \) to be able to quit with honor, further progress is painfully slow. Often the sampling size on \( h \), the \( \Delta h \) intervals, must be shortened beyond all endurance in an effort to find a \( P_1 \) for which \( \varepsilon(P_1) < \varepsilon(P_0) \). Ingenious variations on the basic steepest descent theme have lessened but not removed this difficulty.

J. Marquardt Interpolation in Space \( \tilde{S} \)

The two fitting techniques we have discussed so far are:

(i) differential corrections, which in space \( \tilde{S} \) is based on matrix equation (54); this equation has the component form

\[ \sum_{n=1}^{N} a_{kn}(P) \cdot \Delta \tilde{P}_n = \tilde{b}_k(P) \]  

\[ [k=1,2,\ldots,N] \]

(ii) steepest descent, based on the vector equation (55), which in space \( \tilde{S} \) has the component form

\[ \tilde{P}_1 = \tilde{P}_0 + h \cdot \tilde{b}(P_0) \]}
\[ \Delta p_k = h b_k(P) \] 
\[ [k=1,2,\ldots,N] \]

Comparing these two methods, we note that:

(a) Far from the solution point, the steepest descent technique is superior. It must proceed so as to decrease \( e \), whereas the differential corrections method is under no such compulsion and is likely to lead us into strange pastures.

(b) Close to the solution point, the differential corrections method is superior. It converges rapidly in the very region where the steepest descent technique languishes.

Marquardt* has proposed an interpolation between the two methods: a technique that behaves like the steepest descent when we are far from the solution and like the differential corrections method when we enter a neighborhood in which the higher-order terms in Eqs. (33) and (34) are negligible.

To achieve this interpolation, a positive nondimensional constant \( \lambda \) is added to each diagonal element of the scaled matrix \( A \). That is, the system (56) is replaced by

\[
\sum_{n=1}^{N} \tilde{a}_{kn}(P) \cdot \Delta \tilde{p}_n = \tilde{b}_k(P)
\]  

where

\[
\tilde{a}_{kn} = \begin{cases} 
1 + \lambda & \text{when } k=n \\
\tilde{a}_{kn} & \text{when } k\neq n 
\end{cases}
\]

System (58) is the bedrock upon which the FINLIE fitting process rests. Note the behavior of this system as a function of \( \lambda \):

(a) As \( \lambda \to 0 \), system (58) clearly reverts to the differential corrections system (56).

(b) As \( \lambda \to \infty \), the diagonal terms of system (58) dominate and the system degenerates into \( N \) uncoupled equations of the form

\[ (1 + \lambda) \Delta \tilde{p}_k = \tilde{b}_k \]

*See the Bibliography, part A.
or, since by assumption $\lambda >> 1$,

$$\Delta p_k = \lambda^{-1} \hat{b}_k$$

(60)

Comparing (60) with (57), we see that for large $\lambda$, values, system (58) simulates the steepest descent approach with $h = \lambda^{-1}$. That is, for $\lambda >> 1$, (58) will take us to a new point a rather short distance from the current point $P$ in the direction of the negative gradient.

Marquardt has suggested an algorithm for determining a suitable value of $\lambda$ for each iteration; that is, for each step $P_0$ to $P_1$, $P_1$ to $P_2$, etc., on the path to the desired solution point $P$. This algorithm (with a few very minor "refinements") has been incorporated into FINLIE. The basic scheme is as follows.

For the first iteration, $P_0$ to $P_1$, FINLIE assigns a tentative value to $\lambda$:

(starting $\lambda$) $P_0$ to $P_1 = \lambda_{1A} = 0.001$  (61)

Let $P_{1A}$ denote a candidate for point $P_1$, obtained by solving (58) with $P=P_0$ and $\lambda=\lambda_{1A}$:

$$P_{1A} = \hat{P} + \Delta P(P_0, \lambda_{1A})$$

(62)

The basic test that any point $P$ should pass is that it be an improvement over the current point:

$$\epsilon(P) < \epsilon(P_0)$$

(63)

If $P_{1A}$ satisfies test (63), then FINLIE returns that point to the user as the updated point $P_1$ and is ready to start the next iteration, $P_1$ to $P_2$.

If $P_{1A}$ fails test (63), then FINLIE must take a smaller step in a more propitious direction. This can be accomplished by increasing $\lambda$. That is, FINLIE re-solves system (58) with $P=P_0$ as before, but with $\lambda$ increased to, say,

$$\lambda_{1B} = 10 \lambda_{1A}$$

(64)

(Note that in re-solving the system (58), the elements $\tilde{\sigma}_{kn}$ ($k \neq n$) and $\tilde{\varphi}_k$ do not have to be re-evaluated. They depend only on the current point and thus are evaluated only once each iteration.) The new increment vector for $\lambda_{1B}$ yields the new candidate point.
\[ \hat{p}_{1B} = \hat{p}_0 + \Delta p \left( P_0, \lambda_{1B} \right) \]  

If \( p_{1B} \) satisfies test (63), then FINLIE returns this point to the user; if \( p_{1B} \) fails test (63), then FINLIE increases \( \lambda \) again by a factor of ten, and so on. Sooner or later, an acceptable candidate will be found:

\[ \hat{p}_1 = \hat{p}_0 + \Delta p \left( P_0, 10^n \lambda_{1A} \right) \]  

where \( n \) is zero or a positive integer.

The cost-conscious reader may ask: if \( p_{1A} \) fails test (63), why not skip over a possibly long line of rejected candidates by increasing \( \lambda \) by some factor much larger than ten? This should get us to an acceptable candidate point at once or at least in fewer trials. True, but the general principle is this: the larger the \( \lambda \), the smaller the progress we are making. Hence we don't want FINLIE to use a \( \lambda \) "very much" larger than needed to satisfy test (63). It is not worth the effort to find the optimum \( \lambda \) for each iteration, but by testing after each ten-fold increase in \( \lambda \), FINLIE will not grossly exceed that optimum. (Indeed, a case could be made out for merely doubling \( \lambda \) each time an increase is required.)

The only way in which the second and subsequent iterations differ from the first is in the formula FINLIE uses for determining the starting \( \lambda \) value for the iteration:

\[ \text{(starting } \lambda)_p \text{ to } p_n = 0.1 \times \text{(final } \lambda \text{ value used to produce point } p_{n-1} \text{ in the previous iteration)} \]  

That is, FINLIE always decreases the current value of \( \lambda \) by a factor of ten at the start of each new iteration. This decrease is an essential part of the \( \lambda \) manipulation. When all is going well, FINLIE will have no need to increase \( \lambda \); thus rule (67) will insure that \( \lambda \) goes to zero - and hence that the process approaches the differential corrections technique - as FINLIE approaches the solution point \( \hat{p} \).

A typical set of \( \lambda \) values encountered in the course of some hypothetical fit (not our familiar examples, (3) and (4)) is shown in Table V. The reader can infer from these \( \lambda \) values the fleeting existence of rejected candidate points. Thus, to get from \( P_2 \) to \( P_3 \), FINLIE clearly had to solve system (58) six times: for \( \lambda = 10^{-4} \) (that is, one-tenth the previous \( \lambda \)), \( 10^{-3}, 10^{-2}, 10^{-1}, 10^0 \) and \( 10^1 \) (the \( \lambda \) value that produced a successful candidate). Similarly, to get from \( P_5 \) to \( P_6 \),
<table>
<thead>
<tr>
<th>Iteration</th>
<th>λ value returned by FINLIE at the end of the iteration</th>
<th>No. of times system (58) must have been solved by FINLIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$ to $P_1$</td>
<td>$10^{-2}$</td>
<td>2</td>
</tr>
<tr>
<td>$P_1$ to $P_2$</td>
<td>$10^{-3}$</td>
<td>1</td>
</tr>
<tr>
<td>$P_2$ to $P_3$</td>
<td>$10^1$</td>
<td>6 (for $\lambda=10^{-4}, 10^{-3}, \ldots 10^1$)</td>
</tr>
<tr>
<td>$P_3$ to $P_4$</td>
<td>$10^0$</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$ to $P_5$</td>
<td>$10^{-1}$</td>
<td>1</td>
</tr>
<tr>
<td>$P_5$ to $P_6$</td>
<td>$10^{-1}$</td>
<td>2 (for $\lambda=10^{-2}, 10^{-1}$)</td>
</tr>
<tr>
<td>$P_6$ to $P_7$</td>
<td>$10^{-2}$</td>
<td>1</td>
</tr>
<tr>
<td>$P_7$ to $P_8$</td>
<td>$10^{-3}$</td>
<td>1</td>
</tr>
<tr>
<td>$P_8$ to $P_9$</td>
<td>$10^{-4}$</td>
<td>1</td>
</tr>
<tr>
<td>$P_9$ to $P_{10}$</td>
<td>$10^{-5}$</td>
<td>1</td>
</tr>
<tr>
<td>$P_{10}$ to $P_{11}$</td>
<td>$10^{-6}$</td>
<td>1</td>
</tr>
</tbody>
</table>
FINLIE must have solved (58) twice: for \( \lambda=10^{-2} \) and \( 10^{-1} \). Thereafter, the fitting process seemed to get back on the track and \( \lambda \) decreased steadily. Without Marquardt's \( \lambda \) in the system, it is likely that the search represented by Table V would have gone astray after point \( P_2 \) and come to some abrupt and ignoble conclusion.

K. Convergence Criterion

The question arises: when can the user accept a point returned by FINLIE as being "close enough" to the desired solution? One possible answer is: when FINLIE tells him he can. At the end of each iteration, FINLIE returns to the user a flag whose value indicates whether or not the returned point has satisfied a built-in convergence criterion. (This flag will be discussed in section III(C).)

The convergence criterion installed in FINLIE is as follows. Let \( P_{n-1} \) and \( P_n \) be any two consecutive points returned by FINLIE: the end points of two consecutive iterations. Then FINLIE will signal convergence at point \( P_n \) if and only if

\[
0.99999 < \frac{\varepsilon(P_n)}{\varepsilon(P_{n-1})} < 1
\]  

(68)

The right-hand portion of this double inequality is essentially inequality (63) and hence is always satisfied, thanks to the Marquardt \( \lambda \) feature. The left-hand inequality in (68), however, constitutes an arbitrary definition of convergence: namely, that the percent change in \( \varepsilon \) has dropped below 0.001.

As an example of criterion (68) in action, consider the search summarized by Table II. The values of \( \text{CR}\varepsilon(P_n)/\varepsilon(P_{n-1}) \) listed in the next to last column of that table jump about erratically (always between 0 and 1, of course) before the criterion is satisfied at point \( P_7 \). The sudden transition from the value of CR at \( P_7 \) to its value at \( P_7 \) is not typical. In searches based on more realistically inaccurate measured data, CR will often be close to - and monotonically approach - the value 1 over the final few iterations.

Note that (68) is only a measure of convergence to a local minimum. We have said it before, but it bears repeating: there is no guarantee that the point \( P_n \) satisfying (68) will yield the desired absolute minimum \( \varepsilon \).

Of course, the user need not accept definition (68); he can ignore the FINLIE convergence flag and impose his own convergence test on the data returned by FINLIE after each iteration.
L. Estimated Errors

In addition to computing the estimated standard deviation of the fit:

\[ s = \left[ \frac{e(\hat{P})}{N^2-N} \right]^{1/2} \]  \hspace{1cm} (69)

FINLIE computes \( s_k \), the estimated standard deviation of paramic \( p_k \), \( k=1,2,\ldots,N \).

For linear least-squares, the conventional formula is

\[ s_k = [\Delta_{kk}]^{1/2} s \]  \hspace{1cm} (70)

where

\[ \Delta_{kk} = \text{the k-th diagonal element of the inverse of the unscaled matrix } A. \]

Note that while \( s \) is nondimensional, \( s_k \) has the same dimensions as \( p_k \).

For nonlinear least-square fits, Eq. (70) should be viewed with a healthy suspicion. Indeed, Celmi\( \text{\textsuperscript{33}} \) (Ref. 33 in the Bibliography) points out that even in the linear case, the equation should be applied only in "very limited special cases." Unfortunately, the alternative formula that he develops for \( s_k \) is a rather complicated one involving second-order derivative terms - terms that so far we have managed to avoid. The inclusion of these terms would mean more work not only for FINLIE - which would be acceptable - but for the user, who would have to derive and program some possibly horrendous expressions. The labor here seems out of proportion to its reward, since the "crude" error estimates provided by (70) are usually not all that crude when the search has converged to the proper point. Hence FINLIE returns these estimates to the user and the user is expected to provide his own grain of salt.

(Note that (70) uses only the diagonal elements of the inverse matrix. In some situations, all of the elements of \( A^{-1} \) are useful for error analysis. In these special situations, \( A^{-1} s^2 \) can be regarded as the variance-covariance matrix. However, for nonlinear least squares, we are pushing our luck in making use of the diagonal elements; to try to assign any significance to the off-diagonal elements would really be folly.)

Recall that FINLIE transforms the elements of matrix \( A \) to the scaled space \( S \), Eq. (50), and then replaces the diagonal elements by \( 1+\lambda \). Hence FINLIE actually obtains the paramic error estimates by the relation
\[ s_k = \left( \frac{(1+\lambda) \hat{a}_{kk}}{\hat{a}_{kk}} \right)^{1/2} s \]  

(71)

where

\[ \hat{a}_{kk} = \text{the } k\text{-th diagonal element of the inverse of matrix } \{\hat{a}_{kn}\}, \text{ Eq. (58)} \]

(I felt there should be some compensation in the error estimate formula for the presence of Marquardt's \( \lambda \) in the equations. By a chain of nonrigorous reasoning, I was thus led to insert the \((1+\lambda)\) factor in (71). Since \( \lambda \ll 1 \) for a good fit, \((1+\lambda)\) seems relatively harmless sitting there.)

M. The Composition of FINLIE

So far, the word FINLIE has denoted an apparently monolithic program. Actually, for reasons that seemed persuasive at the time, FINLIE was written as an assemblage of six linked FORTRAN subroutines:

DUBLIN, LONDON, PARIS, BONN, MATINV, MERSO

only one of which - DUBLIN - is called by the user. "FINLIE", then, is merely a convenient name for an ensemble of six subroutines.

[FINLIE is also the name of a permanent file (in Update format) stored on the front end of BRL's Control Data Corporation computer system. (At BRL, this system consists of two linked mainframes: the CYBER 170/Model 173 and the CYBER 70/Model 76.) File FINLIE contains five of the six subroutines: all but MATINV, which is already available from a system library.]

The relationship between

(i) the user's program that calls FINLIE,

(ii) FINLIE

and (iii) the user's subroutine defining his equations,

and the inter-relationship of the six subroutines that constitute FINLIE are all indicated in Figure 1. A vertical bar between two subroutines in the figure indicates that the upper subroutine calls the lower one.

38
Fig. 1. FINLL and the User: A Schematic
All six subroutines of FINLIE are listed in the Appendix. Only four of the six - the four "cities" - were written by the author; the other two (namely, MATINV and MERSO) are general-purpose subroutines to be discussed shortly.* With some minor exceptions in subroutine MERSO (these will be spelled out), the FORTRAN used in FINLIE is a "more or less standard" version of FORTRAN IV (alias FORTRAN 4, alias FORTRAN 66).

Converting FINLIE to a later model FORTRAN - say, FORTRAN 77 - should be relatively uneventful. One possible difficulty is as follows. FINLIE was written for a compiler that automatically retains the values of entities defined within a subroutine but not linked to the calling program. For such a compiler, subsequent calls to the subroutine will find the previous values waiting. However, in FORTRAN 77 the SAVE statement is available for specifying what if anything is to be retained; hence some FORTRAN 77 compilers may not automatically retain local values. In that case, it may be necessary to SAVE the arrays ALPHA and BETA in subroutine DUBLIN.

DUBLIN is the interface between the user and FINLIE. The user must write the FORTRAN program that calls subroutine DUBLIN with the required input data. Hereafter, we will refer to (and think of) the user's calling program as a main program, although it could itself be a subprogram. Each time that DUBLIN is called by this main program, DUBLIN activates the other subroutines of FINLIE, causing one iteration of the search procedure to be carried out. That is, if the main program submits point \( P_{n-1} \) to DUBLIN, DUBLIN will return to the main program the next point \( P_n \). Information and advice on writing the main program and in particular on calling DUBLIN will be given in overwhelming detail in Part III.

Subroutines LONDON, PARIS and BONN are buried within FINLIE, so that their individual purposes should be of little significance to the user. However, the following features of PARIS can be noted from Fig. 1. If the user is fitting a set of differential equations, PARIS calls a numerical integration subroutine MERSO (of which more will be said shortly) and MERSO in turn calls the subroutine - written by the user and arbitrarily labelled ROME in the figure - that defines the differential equations to be fitted. On the other hand, if the user is fitting algebraic or transcendental equations, PARIS calls the user's equation-defining subroutine ROMA directly. Both ROME and ROMA must get additional information from PARIS through the labelled COMMON.

*A reviewer of this paper questioned the implication that DUBLIN, LONDON, PARIS, and BONN are the only "cities" in the sextuplet of subroutines. He went so far as to consult an atlas to see if there is a town, a village, a hamlet or a crossroads by the name of MATINV or MERSO somewhere in the world. Apparently there isn't.
block NAPLES. ROME and ROMA may require additional information from the user's main program, this can be passed through the labelled COMMON block CAIRO. Abundant details on writing ROME and ROMA and on the COMMON blocks will be given in Part III.

Subroutine MATINV is a general-purpose matrix inversion subroutine borrowed intact from the computer library here at BRL. Upon return from MATINV, the input matrix has been replaced by its inverse.

Subroutine MERSO is a general-purpose numerical integration subroutine based on a method proposed by R.H. Merson of Australia. The method is a fourth-order member of the Runge-Kutta family, requiring five function evaluations at each integration step. The subroutine adjusts the integration step size automatically to obtain a predefined accuracy. (All of this is transparent to the FINLIE user.)

The computer library at BRL contains a subroutine MERSON (see References 11 and 32 in the Bibliography) for performing Merson integration. Subroutine MERSO is identical to MERSON with the exception of two statements. Firstly, where MERSON has

\[ \text{DIMENSION T(100), G(100), S(100),} \]

MERSO has increased the three dimensions to 400 each. Secondly, where MERSON has

\[ \text{IF (NT.LE.100) GOTO 100,} \]

MERSO compares NT with 400. The reason for these changes is as follows. The size of the three arrays T, G and S above must equal or exceed

\[ N_5 \equiv N_2 + N_A \]  \hspace{1cm} (72) \]

that is, the number of differential equations \(N_2\) plus the number of influence equations \(N_A\). MERSON requires \(N_5 \leq 100\). In my largest application of FINLIE so far, \(N_5\) exceeded 100 (was, in fact, 368). Hence the minor surgery that altered MERSON into MERSO; the maximum permitted value of \(N_5\) is now 400. This value of 400 appears not only in MERSO but in PARIS, where it is the declared dimensions of arrays \(U\) and \(DU\) (see the Appendix). Hence the user can change the upper limit on \(N_5\) by

(i) changing the dimensions of \(U\) and \(DU\) in PARIS;

and (ii) changing the 400 in the PARIS statement that currently reads:

\[ \text{IF(MS.LE.400) GOTO 24} \]

and (iii) changing the two previously mentioned statements in
MERSO:

\begin{verbatim}
DIMENSION T(400), G(400), S(400)

and IF(NT.LE.400) GOTO 100
\end{verbatim}

Note that a "nonstandard" FORTRAN function appears on the line above statement 410 in MERSO:

\begin{verbatim}
H = SIGN1(H) * HMI
\end{verbatim}

Here SIGN1 is the signum function; if it is not recognized by the user's FORTRAN compiler, the above statement can be replaced by

\begin{verbatim}
IF(H.NE.0.) H=SIGN(1.,H)*HMI
\end{verbatim}

The use of multiple arithmetic and logical assignment statements in MERSO may also be unacceptable to some compilers. In a multiple statement of the form

\begin{verbatim}
VN = \cdots = V2 = V1 = expression,
\end{verbatim}

the assignments are carried out from right to left:

\begin{verbatim}
V1 = expression
V2 = V1, etc.
\end{verbatim}

It should be pointed out after all this exposition on MERSO that when the user is fitting algebraic or transcendental equations rather than differential equations, MERSO is not needed and may be removed from FINLIE.

FINLIE was written for BRL's CDC computer system for which the single precision of real numbers is approximately 14 decimal digits. So far, this has proven adequate for all our FINLIE applications. If the user is working with a machine whose single precision is significantly less than 14 decimal digits, he may have to add some double-precision declarations to the subroutines of FINLIE. One source of trouble is the possibly erratic behavior of $e$ near a minimum, due mainly to round-off noise. Hence a likely candidate for double precision is array GAMMA in subroutine DUBLIN (and its dummy version A in subroutine MATINV). A more complete list of variables that may require double precision includes:

in DUBLIN: EA, EB, EPS, GAMMA
in LONDON: EP, EPS, RSQ
in PARIS: RM, RSQ
in MATINV: A, T1

In extreme cases, the user can simply double-precision everything in
sight; this may be inefficient in terms of storage, but it could save wear and tear on the user.

III. OUTSIDE FINLIE: WHAT THE USER MUST DO FOR FINLIE

Assuming that the FINLIE user is given a set of equations of the form (1) or (2) - or equations that can be put into one of those forms - the first task the user must perform is to derive the associated influence equations, as indicated in parts C and D of Section II. The next task is to write a FORTRAN subroutine defining all these equations - the original set and the influence equations - in a manner acceptable to FINLIE. The rules for constructing this subroutine are slightly different for sets (1) and (2). (We assume in what follows that the reader has some familiarity with - though he need not be an expert in - some version of FORTRAN equivalent to or newer than FORTRAN IV.)

A. ROME: The User's Subroutine for Fitting Differential Equations

The first three statements of ROME have the form:

```
SUBROUTINE ROME(N5,XE,U,DU)
DIMENSION U(N5), DU(N5)
COMMON/NAPLES/PAR(40), FLAG(60)
```

It should be noted that the only name in the above three statements that the user is not allowed to change is NAPLES. All other names, including ROME, may be replaced by other legal FORTRAN names of the user's choice. (Of course, the distinction between integer and real names should be maintained.)

ROME is called by MERSO (see Figure 1) and hence the nature of the four arguments of the SUBROUTINE ROME statement has been decreed by MERSO. The first three arguments are input to ROME (from MERSO):

- **N5** = the number of equations to be defined in ROME: N2 (first-order differential equations) plus NA (influence equations). Thus for sample set (3), the value of N5 is 2 + 8 = 10. Note, however, that this argument is an integer name, not an integer constant. As certain arrays are currently dimensioned, N5 cannot exceed 400 (see the pertinent remarks in section II(M)).
XE = x, the independent variable value at which the N5 equations are to be evaluated. The argument, of course, must be a real name, not a real constant. If the independent variable does not appear in any of the equations, then argument XE will not be used in the body of subroutine ROME.

U = the vector of N2 dependent variables and NA influence coefficients, where

\[
\begin{align*}
    y_j &= U(J), \\
    D_{jk} &= U(J + k*\text{N2}) \\
    &\quad [J=j=1,2,\ldots,\text{N2} \\
    &\quad k=k=1,2,\ldots,\text{N2}] 
\end{align*}
\]

Thus for sample set (3), where N2=2 and N23=4, we have

\[
\begin{align*}
    U(1) &= y_1 \\
    U(2) &= y_2 \\
    U(3) &= D_{11} = \frac{\partial y_1}{\partial y_1} \\
    U(4) &= D_{21} = \frac{\partial y_2}{\partial y_1} \\
    U(5) &= D_{12} = \frac{\partial y_1}{\partial y_2} \\
    U(6) &= D_{22} = \frac{\partial y_2}{\partial y_2} \\
    U(7) &= D_{13} = \frac{\partial y_1}{\partial a_1} \\
    U(8) &= D_{23} = \frac{\partial y_2}{\partial a_1} \\
    U(9) &= D_{14} = \frac{\partial y_1}{\partial a_2} \\
    U(10) &= D_{24} = \frac{\partial y_2}{\partial a_2} 
\end{align*}
\]

The final argument of ROME is an output (to MERSO):

DU = the derivative vector at the current value XE of the independent variable, where

\[
DU(J) = \frac{dU(J)}{dx} \quad [J=1,2,\ldots,N5]
\]

Additional input to ROME comes from PARIS via the labelled COMMON block NAPLES. The one hundred elements of the NAPLES block are as follows:

PAR = a vector of the current values of the N3 parameters (not paramics), where N3 ≤ 40. For sample set (3),
PAR(1) = c_1
PAR(2) = c_2
and the remaining 38 elements of PAR are undefined.

FLAG = a vector of N23 flags (N23 \(\leq 60\)) associated with the
N23 single-round paramic set Q, Eq. (12). That is,
the first N2 elements of FLAG are associated with the
N2 initial conditions and the remaining N3 elements
of FLAG are associated with the N3 parameters. The
value of FLAG(J) is

(i) zero if the value of the corresponding
paramic q_j is fixed;
or (ii) 1.0 if the value of q_j is to be adjusted
by the fitting process.

For sample set (3), we have

FLAG(1) = flag for \(y_{10}\)
FLAG(2) = flag for \(y_{20}\)
FLAG(3) = flag for \(c_1\)
FLAG(4) = flag for \(c_2\)
and the remaining 56 flags are undefined.

Note that PAR and FLAG are inputs to ROME from FINLIE; when
writing ROME, the user assumes that the two arrays already contain
their proper values. In the case of the initial condition flags,
these values may change from round to round. For example, in our
tri-round situation, we might decide to make a computer run with \(y_{10}\)
for round E1 and \(y_{20}\) for round E3 fixed at specified values. Then

FLAG(1) = 0.0, 1.0, 1.0
FLAG(2) = 1.0, 1.0, 0.0

for rounds E1, E2 and E3, respectively. FINLIE will automatically
change the values of FLAG(1) and FLAG(2) to match the round whose
measured data is currently being fitted. Of course, FINLIE can't
guess what the user wants to do; it must be told. FINLIE can only
define PAR and FLAG on the basis of certain inputs given to it by the
user's main program. These inputs will be discussed in section III(C).

The dimensions of PAR and FLAG are arbitrary to this extent: they
can be changed in ROME if the user is willing to make all the
associated changes in FINLIE. To save space, I leave the nature of
such changes as an exercise for the interested reader. The simplest
course is to make no changes if \( N2 < 40 \) and \( N23 < 60 \).

After writing down the first three statements of ROME, the user is ready to encode the body of the subroutine: the statements defining the \( N5 \) elements of output array DU. Consider, for example, system (3). For convenience, we repeat here the original equations:

\[
\begin{align*}
y_1' &= \frac{1}{y_2} \\
y_2' &= -c_1(1+c_2y_2)y_2
\end{align*}
\]

and the associated influence equations (Table III-A):

\[
\begin{align*}
(D_{11})' &= -\frac{D_{21}}{y_2^2} \\
(D_{21})' &= -c_1(1+2c_2y_2)D_{21} \\
(D_{12})' &= -\frac{D_{22}}{y_2^2} \\
(D_{22})' &= -c_1(1+2c_2y_2)D_{22} \\
(D_{13})' &= -\frac{D_{23}}{y_2^2} \\
(D_{23})' &= -c_1(1+2c_2y_2)D_{23} - (1+c_2y_2)y_2 \\
(D_{14})' &= -\frac{D_{24}}{y_2^2} \\
(D_{24})' &= -c_1(1+2c_2y_2)D_{24} - c_1(y_2^2)
\end{align*}
\]

For these equations, a likely version of subroutine ROME is given in Table VI.
Table VI. Subroutine ROME for System (3)

SUBROUTINE ROME (NS,XE,U,DU)
DIMENSION U(NS),DU(NS)
COMMON /NAPLES/ C1,C2,BLANK(38),FLAG(60)

V = U(2)
DU(1) = 1./V
A1 = C2*V
DU(2) = -C1*(1. + A1)*V
A2 = DU(1)**2
A3 = C1*(1. + A1 + A1)

IF (FLAG(1) .EQ. 0.) GOTO 10
DU(3) = -A2*U(4)
DU(4) = -A3*U(4)

10 IF (FLAG(2) .EQ. 0.) GOTO 20
DU(5) = -A2*U(6)
DU(6) = -A3*U(6)

20 IF (FLAG(3) .EQ. 0.) GOTO 30
DU(7) = -A2*U(8)
DU(8) = -A3*U(8) - (1. + A1)*V

30 IF (FLAG(4) .EQ. 0.) GOTO 40
DU(9) = -A2*U(10)
DU(10) = -A3*U(10) - C1*V*V

40 RETURN
END

Note that in COMMON/NAPLES/ I opted to write the forty-element parameter set in the form

C1, C2, BLANK(38)

since only the first two of the forty elements have any meaning. I could just as well have written PAR(40) in the COMMON statement and used PAR(1) and PAR(2) instead of C1 and C2 in the body of the subroutine. Note also that I dimensioned FLAG as 60 even though the last 56 elements are meaningless. This was a courtesy to our CDC
FORTRAN compiler, which likes all COMMON blocks of the same name (NAPLES in this case) to have the same length. The compiler doesn't insist when you break this rule, but it comments on your bad form.

Because ROME will be called many times by MERSO during the course of the numerical integration, the user should take the time to make ROME as efficient as practicable. For large and labyrinthian systems of equations, a worthy ROME isn't built in a day.

One of the aids to efficiency in ROME is the FLAG vector. Note that if any paramic value is fixed during a computer run (that is, if the associated flag value is zero), the influence equations for that paramic need not be calculated. Hence the FLAG vector can - and in my opinion should - be used as indicated in Table VI to avoid these unnecessary calculations. The general rule is that if FLAG(J) is zero, then ROME need not evaluate DU(LA) through DU(LB), where

\[ \begin{align*}
LA &= (J \times N2) + 1 \\
LB &= (J \times N2) + N2 = LA + (N2-1).
\end{align*} \]

Of course, if the user is convinced that he will never, ever want to hold fixed the value of some paramic, he can omit the corresponding IF-statement from ROME.

Some systems of equations may involve constants whose values are always fixed (that is, never adjusted by FINLIE) and yet these values may change from run to run. It would be possible - but not too bright - to handle such a constant as a fixed parameter: a parameter whose associated flag is always zero. A better approach is to pass any such constant directly from the user's main program to ROME through a new labelled COMMON block (see block CAIRO in Figure 1). Of course, if a constant will never change from run to run, it need only be defined within ROME.

A final, rather minor comment: Sample set (3) is one of those cases where the input argument XE is not used in the body of subroutine ROME, simply because the independent variable does not appear explicitly in the N5 equations of this example.

B. ROMA: the User's Subroutine for Fitting Algebraic or Transcendental Equations

Many of the comments in the previous section concerning ROME apply to ROMA as well. Hence, if the reader has skipped over that section because his interest in fitting differential equations is minimal, he may have missed something noteworthy. Or possibly not.

The first three statements of ROMA have the form:
SUBROUTINE ROMA (COND, XO, XE, U)
DIMENSION COND(n2), U(n5)
COMMON/NAPLES/PAR(40), FLAG(60)

The first three arguments in the SUBROUTINE statement are inputs (from PARIS):

COND = a vector of N2 current initial condition values. For sample set (4),

\[ \text{COND}(1) = y_{10} \]
\[ \text{COND}(2) = y_{20} \]

For multi-round situations, the initial conditions change with the round as well as with the current state of the fitting process. FINLIE supplies the proper COND vector to ROMA automatically.

XO = \( x_0 \), the independent variable value at which the initial conditions apply. (This one value must apply to all rounds.)

XE = \( x \), the independent variable value at which the equations are to be evaluated.

The final argument, U, is an output vector defined exactly as in the previous section for subroutine ROME.

In the DIMENSION statement, n2 and n5 denote the values of N2 and N5, respectively. (Actually, on the CDC system and on most other computers, a one-dimensional argument array in a subroutine need not be declared at its maximum size; the value 1 is adequate.)

The labelled COMMON block NAPLES brings to ROMA the arrays PAR and FLAG, defined in the previous section.

The body of subroutine ROMA consists of the statements defining the needed elements of array U. Consider, for example, system (4). For convenience we repeat here the original equations (4) and the needed influence equations (Table III(B)):

\[ y_1 = y_{10} - c_2 (x-x_0) + \frac{(b/c_1)(u-1)}{u} \]
\[ y_2 = \frac{(bu - c_2)^{-1}}{u} \]
\[ D_{11} = 1. \]
\[ D_{12} = -\frac{(u-1)/(c_1 y_{20})^2}{u} \]
\[ D_{13} = \frac{b}{c_1^2} [1-u+c_1(x-x_0)u] \]
\[ D_{14} = \frac{u-1}{c_1} -(x-x_0) \]

where
\[ u = \exp \left[c_1(x-x_0)\right] \]
\[ b = (y_{20})^{-1} + c_2 \]

For these equations, a likely version of subroutine ROMA is given in Table VII.

<table>
<thead>
<tr>
<th>Table VII. Subroutine ROMA for System (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBROUTINE ROMA (COND,XO,XE,U)</td>
</tr>
<tr>
<td>DIMENSION COND(2),U(10)</td>
</tr>
<tr>
<td>COMMON /NAPLES/ C1,C2,BLANK(38),FLAG(60)</td>
</tr>
<tr>
<td>T0 = COND(1)</td>
</tr>
<tr>
<td>VO = COND(2)</td>
</tr>
<tr>
<td>XA = XE - XO</td>
</tr>
<tr>
<td>Z = EXP(C1*XA)</td>
</tr>
<tr>
<td>B = C2 + 1./VO</td>
</tr>
<tr>
<td>U(1) = T0 - C2<em>XA + B</em>(Z - 1.)/C1</td>
</tr>
<tr>
<td>U(2) = 1./(B*Z - C2)</td>
</tr>
<tr>
<td>IF (FLAG(1), ,NE. 0.) U(3) = 1.</td>
</tr>
<tr>
<td>IF (FLAG(2), ,NE. 0.) U(5) = (1. - Z)/(C1*VO**2)</td>
</tr>
<tr>
<td>IF (FLAG(3), ,NE. 0.) U(7) = B*(1. - Z + C1<em>XA</em>Z)/(C1**2)</td>
</tr>
<tr>
<td>IF (FLAG(4), ,NE. 0.) U(9) = (Z - 1.)/C1 - XA</td>
</tr>
<tr>
<td>RETURN</td>
</tr>
<tr>
<td>END</td>
</tr>
</tbody>
</table>

As discussed in section II(D), FINLIE does not require expressions for the influence coefficients \( D_{jk} \) when \( j \) is greater than \( N1 \). Hence in this sample ROMA, where \( N1=1 \), the \( D_{2k} \) equations (namely, the equations for \( U(4), U(6), U(8) \) and \( U(10) \)) are simply omitted from the subroutine.

As with ROME in the previous section, the FLAG array in ROMA is used to avoid calculating \( D_{jk} \) when the value of paramic \( q_k \) is fixed. Also as with ROME, any needed "changeable constants" can be passed directly from the user's main program to ROMA through, say, the labelled COMMON block CAIRO.
C. Calling Subroutine DUBLIN

After the user has written his subroutine defining the equations to be fitted, his next task is to write a program unit - we assume a main program - that utilizes FINLIE. Before discussing this main program as a whole, we will concentrate on one statement within that main program: the CALL DUBLIN statement.

This statement is the link between the user and FINLIE. It can be written in the form

```
CALL DUBLIN (ROME,NF,N1,N2,N3,N7,N8,NR,NM,XO,
 X,Y,F,NW,P,RL,NC,YC,R,RS,EPS,
 SIG,EK,NS)
```

where all integer names happen to start with the letter N. The first fourteen of the twenty-five arguments are inputs.

1. **ROME** is the name of the subroutine (written by the user) that defines the equations to be fitted (See sections III(A-B)).

The values of the remaining thirteen input arguments must be established in the user's main program before DUBLIN is called. These values will not be changed by FINLIE; hence actual values rather than names may be used for arguments [2] through [8], [10] and [14] below.

2. **NF** is a flag that indicates the nature of the equations to be fitted:

   \[ NF=0 \text{ if the fitting equations are algebraic or transcendental (System (2));} \]

   \[ NF=1 \text{ if the fitting equations are differential equations (System (1)).} \]

3. **N1** is the number of measured dependent variables in the system, where

   \[ 1 < N1 < 10 \]  \hspace{1cm} (75)

   (The upper bound on N1 - and the upper bounds indicated for some of the other arguments defined below - can be increased only by delving into FINLIE.) N1 must have the same value for each round; FINLIE insists that the same dependent variables be measured for each round used in the fitting process.

4. **N2** is the total number of dependent variables in the system, where
\[ N_1 \leq N_2 \leq 20 \] \hspace{1cm} (76)

[5] \textbf{N3} is the maximum number of parameters (not paramics) whose values can be determined from the fit, where

\[ 0 \leq N_3 \leq 40 \] \hspace{1cm} (77)

I use the word "maximum" above because the actual number of parameters to be determined in the course of a computer run may be less than N3. The user specifies (by argument F, to be discussed below) which, if any, of the parameters and initial conditions are to be held fixed at their input values and which are to be adjusted by FINLIE during the run. Input N3 is the total number of parameters: those to be adjusted plus those held fixed. (If input N3 is zero - the lower limit in inequality (77) - then presumably there is at least one initial condition to be determined; otherwise there would be no reason for running the program.)

[6] \textbf{N7} is the number of rows declared in the user's main program for the two-dimensional arrays Y, W and R defined below as arguments [12], [15] and [20], respectively. As we will see, these three arrays serve as N1 by N4 matrices. At first glance, then, it might seem that N7 = N1. However, the user may not want to restrict his main program DIMENSION statement to the current values of N1 and N4. It is often more convenient to dimension arrays at their largest anticipated sizes. For example, in our recurring case where N1 = 1 and N4 = 16, the user might want to dimension arrays Y, W and R as, say (2, 50) rather than (1, 16). FINLIE will go along with this sort of thing, but it wants to be told about it. Thus if the user dimensions Y, W and R as (2, 50), he must set N7 equal to 2. In general, then, N7 \geq N1. (The declared column dimension for the three arrays - say, 50 - is not needed by FINLIE. The declared row dimension is sufficient - assuming the computer stores matrices in the usual way, that is, by columns - to maintain notational row-column agreement between calling program and subroutine. Neither is FINLIE interested in the declared dimensions of its vector arguments.*)

*) In FINLIE, I have declared 1 as the last (right-most) dimension of subroutine dummy argument arrays. This is fairly common FORTRAN 4 practice, but FORTRAN 77 prefers an asterisk: \( Y(N7, *) \) instead of \( Y(N7, 1) \).
[7] **N8** is the number of rows declared in the user's main program for the two-dimensional array YC defined below as argument [19]. Array YC serves as an N2 by N4 matrix; hence N8 > N2. (See the comments for argument [6] above.)

[8] **NR** is the number of data rounds to be considered simultaneously, where

\[ 1 \leq NR \leq (60-N3)/N2 \]  \hspace{1cm} (78)

The right-half of this double inequality may seem a rather strange condition to spring upon the reader. Until now, no limit has been implied on the number of rounds. The basic condition (somewhat concealed in (78)) is

\[ N \leq 60 \]  \hspace{1cm} (79)

where N is the total number of paramics, (NR x N2) + N3. Condition (79), like the limits on N1, N2 and N3, is a result of arbitrary DIMENSION decisions that had to be made when constructing FINLIE. Since N is not itself an input to DUBLIN, I have simply converted (79) to the equivalent form (78). By satisfying (78), the user can be sure that (79) is also satisfied. For sample set (3) or (4) we must have NR < (60-2)/2=29. For the associated data of Table I, we have NR=3, well below the maximum permitted. Recall that the data for an individual round solely determine the initial conditions for that round, but combine with the data from all the other rounds to determine the parameters.

[9] **NM** is a vector of NR elements, where

\[ NM(J) = \text{the number of data points } R_m \text{ (that is, the number of independent variable values } x_m \text{ at which measurements were taken) for the } J\text{-th round.} \]

Thus for the sample data of Table I, the user's main program must set

\[ NM(1) = 5 \]
\[ NM(2) = 6 \]
\[ NM(3) = 5 \]

FINLIE determines N4, the total number of data points, by summing the NM components.
The user must insure that $N_4$ satisfies the inequalities

$$N < N_4 < 1000$$
$$N_4 \times N_5 < 10000$$

and that

$$N_1 \times (\text{MAX.ELEMNT OF NM}) < 200$$
$$N_2 \times (\text{MAX.ELEMNT OF NM}) < 400$$

Again, these restrictions are the result of arbitrary DIMENSION statements in FINLIE.

[10] $X_0$ is the independent variable point $x_0$ at which all initial conditions apply. The same $x_0$ must apply to all rounds.

[11] $X$ is a vector of the $N_4$ independent variable values $x_m$ at which measurements were taken. The first $N_M(1)$ values in $X$ are the first-round values, in increasing order:

$$X(M-1) < X(M), \ M = 2, 3, \ldots N_M(1)$$

The next $N_M(2)$ values of $X$ are the second-round values, also in increasing order among themselves:

$$X(M-1) < X(M), \ M - N_M(1) = 2, 3, \ldots N_M(2)$$

and so on. For the Table I data, we have

$$X(M) = x_m, \ m = 1, 2, \ldots 16.$$ 

[12] $Y$ is an $N_1$ by $N_4$ matrix of measured dependent variable values, where

$$Y(I, M) = \text{the measured value of } y_i \text{ at } X(M)$$

For the Table I data, we have $Y(1, M) = \bar{y}_{1m}, \ m = 1, 2, \ldots 16.$

[13] $F$ is a vector of $N$ flags associated with the paramic point $P$ (argument [16] below), where
\[ F(J) = 0.0 \text{ if the input value of } P(J) \text{ is to be held fixed;} \]
\[ = 1.0 \text{ if the input value of } P(J) \text{ is to be adjusted by the fitting process.} \]


\[ NW = 0 \text{ if the user's weights (already stored in matrix } W \text{) are to be used by FINLIE;} \]
\[ = 1 \text{ if all weights are unity (in which case, the user need not store } 1.0' \text{s in matrix } W \text{ before calling DUBLIN).} \]

Recall the comments in the vicinity of Eqs. (8) and (9) regarding weights. The important point is that the "easy" way out - assigning unit weights, merely by setting NW=1 - will often lead to a poor fit. Give some minimum consideration to the possibility of unequal uncertainties in the measurements, particularly when more than one variable has been measured (\( N_1 > 1 \)).

The fifteenth argument of DUBLIN may or may not be defined by the user before DUBLIN is called:

[15] W is the \( N_1 \) by \( N_4 \) matrix of weights associated with input Y (argument [12] above). The user has a choice to make. If each of the \( N_1 \) by \( N_4 \) measurements in matrix Y can be assigned unit weight; that is, if

\[ W(I,J) = 1.0 \]

then the user need not define the W array. Simply set NW (argument [14] above) to 1. If, on the other hand, the user decides that one or more of the weights must differ from 1, then the user must define the entire array, subject to the conditions that each weight be nonnegative and that

\[ [W(I,J)]_d = [1./Y(I,J)^2]_d \]

See the comments near Eqs. (8) and (9).

The next three arguments of DUBLIN are input/output. That is, the user must define them before the first CALL DUBLIN statement, but FINLIE will change their values.
P is the current N-dimensional paramic point $P$, Eq. (5), where

$$P(1), \ldots, P(N2) = \text{first round initial conditions},$$

$$P(1+N2), \ldots, P(2*N2) = \text{second round initial conditions},$$

$$\ldots$$

$$P(1+(NR-1)*N2), \ldots, P(NR*N2) = \text{last round initial conditions},$$

$$P(1+NR*N2), \ldots, P(N) = \text{the parameters}.$$

Thus for sample system (3) or (4) and for tri-round data, the eight elements of $P$ are given by Eq. (10). Clearly, $P$ is the essential argument in the CALL statement; the other arguments play a necessary but supportive role. As indicated in Part I of this report, an effort should be made to find suitable starting values for the elements of $P$. Not all first estimates will lead to the right answer. Each time the program returns from DUBLIN, array $P$ will contain an updated point. More precisely, the first call to DUBLIN is a special situation and $P$ is unchanged upon return. Thereafter, each call serves to update $P$. For more on this first call, see argument [18] below. In general, then, each DUBLIN call after the first advances $P$ one step on the road to the solution. DUBLIN should be called repeatedly (say, in a DO-loop) until convergence is achieved. Not all elements of $P$ will necessarily change with the iteration. If input $F(J)$ is zero (see argument [13] above), then the original, user-assigned value of $P(J)$ will be maintained no matter how many times DUBLIN is called.

RL is a Marquardt argument. Before DUBLIN is called the first time,

(i) Set $RL = 0.0$ if the Marquardt algorithm is to be omitted from the fitting process (that is, if the user wants FINL'E to fit by differential corrections, Eq. (56), rather than by Marquardt interpolation, Eq. (58)). In this case, $RL$ will remain at zero.

(ii) Set $RL = 1.0$ if the Marquardt algorithm is to be used. Upon the first return from
DUBLIN, RL will have the "starting" λ value of 0.01. (On subsequent calls to DUBLIN, the input value of RL is immediately divided by ten; hence the true starting value of λ is 0.001, as indicated in Eq. (61)).

Upon the second and subsequent returns from DUBLIN, RL will have the value of λ used to obtain the point returned in array P. Note that since FINLIE changes RL, a name (not the value 1.0) must be used in the CALL list.

[18] NC is a "first call" flag. The user must set NC=0 initially. This value alerts FINLIE to the fact that it is being called for the first time. FINLIE behaves differently on this first call than it does on all subsequent calls. In particular, on the first call, FINLIE

(i) sets all elements of argument W to unity if NW=1;
(ii) sets argument RL to 0.01 if the input RL is 1.0 (that is, if Marquardt's algorithm is to be used);
(iii) determines the number of parameters to be adjusted (the total number minus the number of parameters held fixed) and stores this value back in argument NC (hence use a name, not the integer zero, for the "first call" flag in the CALL list);
(iv) evaluates the next five arguments in the CALL list (YC, R, RS, EPS and SIG, all described below) at the input point P₀.

Note that FINLIE does not update the input point P₀ on this first call: P₀ goes in and P₀ comes back. The parameters are updated only on the second and subsequent calls.

The remaining seven arguments of DUBLIN are outputs, evaluated at the current value of P.

[19] YC is an N2 by N4 matrix of computed dependent variable values, based on the current point P, where

\[ YC(J,M) = \text{the computed value of } y_j \text{ at } X(M) \]

Thus for the Table I data in our examples,
\[ YC(1,M) = y_1(x_m, P) \]
\[ YC(2,M) = y_2(x_m, P) \]

\[ m = 1, 2, \ldots, 16 \]

When fitting differential equations, FINLIE obtains the YC values by numerical integration of system (1); when fitting algebraic or transcendental equations, FINLIE obtains YC directly from the equation set (2).

[20] \( R \) is an \( N_1 \) by \( N_4 \) matrix of residuals of the fit, where
\[ R(I, M) = Y(I, M) - YC(I, M) \]

[21] \( RS \) is a vector of \( N_1 \) nondimensional error measures associated with the \( N_1 \) measured dependent variables, where
\[ RS(I) = \text{that part of } \varepsilon \text{ (see Eq. (7) and argument [22] below) that can be attributed to the fit on } y_i \]
\[ = \sum_{M=1}^{N_4} W(I, M) [R(I, M)]^2 \]

[22] \( EPS \) is \( \varepsilon(P) \), the nondimensional sum of the weighted squares of the residuals of the fit (Eq. (71)), where
\[ EPS = \sum_{I=1}^{N_1} RS(I) \]

If the Marquardt feature is being used (see argument [17]), then after the first call, DUBLIN should return an EPS no greater than the input EPS.

[23] \( SIG \) is the estimated standard deviation of the fit (Eq. (69)), where
\[ SIG = \left[ \frac{EPS}{N_4 - N} \right]^{1/2} \]

[24] \( EK \) is a vector of crude estimates of the errors in the \( N \) parameters of point \( P \), where
\[ EK(K) = \text{the estimated standard deviation in paramic } P(K) \]
\[ s_k \text{ as defined in Eq. (71)} \]

[25] \(NS\) is a convergence flag. Before returning to the user's main program, \(FINLIE\) will set

\(NS=0\) if the process has not yet converged by criterion (68), but there is still hope. \(FINLIE\) is saying in effect, "Nothing obvious has gone wrong yet so give \(DUBLIN\) another call."

\(NS=1\) if all output arguments (except this one) are invalid. Usually this happens when some input argument is invalid. (\(FINLIE\) performs a few simple checks to spot invalid inputs.) If \(DUBLIN\) returns an \(NS\) value of 1, the main program should take some special action (e.g., STOP).

\(NS=2\) if the latest iteration has satisfied convergence criterion (68). If the user is willing to accept this criterion, his main program should stop calling \(DUBLIN\) when \(NS=2\). If the user is imposing some more stringent convergence criterion of his own, he should regard \(NS=2\) as having the same meaning as \(NS=0\).

To illustrate the use of these twenty-five arguments, consider our sample systems (3) and (4) with three-round data. Assume that in the calling program, arrays \(Y\), \(W\), \(R\), and \(YC\) have been dimensioned as \((2,50)\). Then for system (3) and subroutine \(ROME\), we can write

\[
\text{CALL DUBLIN}(ROME,1,1,2,2,2,3,NM,0.0,X,Y,F,1,W,P,RL,NC, YC,R,RS,EPS.SIG,EK,NS)
\]

For system (4) and subroutine \(ROMA\), only the first two arguments above are changed:

\[
\text{CALL DUBLIN}(ROMA,0,...)
\]

D. Writing the Program that Calls \(DUBLIN\)

In this final section, a typical main program for utilizing \(FINLIE\) is broken down into six steps. Some of these steps are essential, others are optional.

**Step (1).** Dimension all ten arrays appearing in the \texttt{CALL DUBLIN} statement:

\[
\text{DIMENSION } NM(nr), X(n4), Y(n1,n4), F(n), W(n1,n4), P(n), YC(n2,n4), R(n1,n4), RS(n1), EK(n)
\]
where small-letter dimensions above denote constants no less than the values of the corresponding capital-letter names. That is, \( n_r \geq N_r \), etc. As we have mentioned, it is often useful to dimension arrays larger than their current working sizes. For example, in our tri-round test cases (3) and (4), we might write:

```
DIMENSION NM(5), X(50), Y(2,50), F(8), W(2,50), P(8), YC(2,50),
R(2,50), RS(2), EK(8)
```

This would allow for up to five rounds \((n_r=5)\), fifty measurement points \((n_4=50)\) and two measured variables \((n_1=2)\). Note that the values given to the row sizes \(n_1\) and \(n_2\) in this DIMENSION statement become the values of arguments \(N_7\) and \(N_8\) when DUBLIN is called. On the other hand, the dimensions allotted above to the vector arguments and to the columns of the matrix arguments are of no interest to FINLIE.

**Step (2).** Declare in an EXTERNAL statement the user subroutine whose name will be passed to DUBLIN. Thus for sample set (3) and the corresponding ROME (Table VI), we would write

```
EXTERNAL ROME
```

and similarly for set (4) and ROMA.

**Step (3).** Establish initial values for seven DUBLIN arguments:

```
NM, X, Y, F, P, RL, NC
```

and if necessary, for an eighth argument: \( W \). There is no standard coding for obtaining the values of these arguments; the technique will vary with the situation. For example, initial estimates for array \( P \) might be read in at this stage, or they might be obtained by calling some subroutine whose sole purpose is to derive adequate estimates from the data. For simplicity, let's assume that in our main program for sample set (3) or (4),

(a) the arrays \( NM, X, Y, F \) and \( P \) are read in;

(b) \( RL \) and \( NC \) are defined explicitly:

```
RL = 1.0
NC = 0
```

(c) array \( W \) is not defined (since argument \( NW \) will be 1 in the CALL statement).

Note that the values of the remaining nine input arguments:

```
NF, N1, N2, N3, N7, N8, NR, XO, NW
```
can be established in the CALL DUBLIN statement itself.

Step (4). Write column headings for everything of interest that will be determined at the end of each iteration. Of course, "interest" is subjective. One user may want a detailed print-out of the progress from $P_0$ to $P$; a less inquisitive user may care only for what pertains to the final, converged point. Personally, for each iteration, I like to see:

(a) the iteration number $i$ ($i=0,1,2,...$)

(b) the $N$ elements of point $P_i$

(c) the value of Marquardt's $\lambda$ required to produce $P_i$

(d) the residual function $\epsilon(P_i)$ and/or the standard deviation of the fit $s(P_i)$.

These desiderata, then, determine my column headings. (Of course, if what I want to see cannot be conveniently spread across a single output page, then some of the results of each iteration have to be saved - by storing them in additional arrays - so that they can be printed later on a second page.)

Step (5). Program the DO-loop that calls DUBLIN. For our sample set (3), we might write:

```fortran
DO 60 K=1,26
   CALL DUBLIN (ROME,1,1,2,2,2,2,3,NM,0.0,X,Y,F,
                 1   1,W,P,RL,NC,YC,R,RS,EPS,SIG,EK,NS)
   NPOINT=K-1
   WRITE(6,100)NPOINT,PRL,SIG
   IF (NS-1) 60,70,80
60 CONTINUE
   WRITE (6,101)
   C --------The above is a warning that the process has
   C --------failed to converge in 25 iterations.
   GOTO 80
70 WRITE (6,102)
   C --------The above is a warning that something is wrong.
   STOP
80 CONTINUE
```

In the above code, DUBLIN will be called until output argument NS equals 1 or 2, or until the DO-loop variable K exceeds 26, whichever occurs first. (The limit 26 - that is, 25 iterations - is arbitrary; 1 is not enough, 1010 is too many.) After each iteration, we obtain a print-out - presumably under the proper column headings - of NPOINT (the number of iterations), the $N$ elements of the current point $P$, and
finally the \( \lambda \) and \( s \) values at the current point. The first line of this print-out, where \text{NPOINT}=0, gives the initial estimated values of the paramics. If \text{NS}=1 at the end of any iteration, the program stops; otherwise the program moves eventually to statement 80. Note: the principal results of the fitting process are the final printed values of the \( N \) paramics. All else is in a sense window-dressing.

Step (6). Write anything else of interest. My usual scheme is as follows:

(a) aligned under the final paramic values (but with one line skipped for clarity), I write the corresponding crude error estimates contained in array \text{EK}. (If flag array \text{F} is of interest, the elements of \text{F} can be written on the next line, again aligned under the corresponding paramic values.)

(b) on a new output page, I write the values stored in arrays \( X, Y, YC, R \) and (possibly) \( W \), one line for each \( X \) value. (In multi-round fits, I skip a line for clarity at the end of the data for each round.)

(c) wherever convenient, I write the suitably labelled values of some or all of the following:

\[ N1, N2, N3, N4, N, N_C, N_M, N_R, N_W, R_S, X_O \]

IV. SUMMARY

The recent patter of tiny details has very likely blurred the big picture. To review, then, assume that the reader has a problem reducible to fitting a set of equations of the form (1) or (2) to measured data. Further assume that this reader--an adventurous spirit--decides to use FINLIE to solve the problem. Then this invoker of FINLIE must:

(a) derive the related set of influence equations (Section II, C or D);

(b) write a FORTRAN subroutine that lists the original equations and the related influence equations (Section III, A or B);

(c) write a FORTRAN main program (Section III, D) that will:

(i) furnish adequate initial estimates of the parameters and initial conditions;

(ii) specify which, if any, of these estimates are to be adjusted by FINLIE;

(iii) assign weights to the measurements (if the weights are not all equal);

(iv) call subroutine DUBLIN (Section III, C) in a DO-loop;
(d) submit the entire program (main, FINLIE and equation-defining subroutine) to the computer and ponder the ensuing output.

This output will take one of four forms, listed in decreasing order of desirability:

1. convergence to the right answer;
2. failure to converge in the specified number of iterations (sometimes achieving an apparent oscillation about an answer);
3. divergence (the program crashes);
4. convergence to the wrong answer.

Result (1) above--convergence to the right answer--should prevail when all of the following hold:

(a) the measured data are a representative sample of the total behavior they are meant to define. (An elementary violation would be measurements taken every \( \tau \) seconds on a periodic variable of period \( \tau \).)

(b) the measured data are free of gross errors.

(c) "least squares" is a suitable fitting criterion. (This implies that the measurement errors possess certain statistical traits; however, the degree to which the errors must possess these traits in order to be considered amenable to least squares is a matter of judgment.)

(d) the fitting equations with their associated parameters are appropriate for describing the measured events.

(e) the initial paramic estimates are not too far from the right answer. (What constitutes "too far" varies with the nature of the fitting equations and the measured data.)

V. ACKNOWLEDGEMENTS

We have already acknowledged our debt to Marquardt, whose algorithm [see References 1 to 7 in the Bibliography] has been incorporated into FINLIE. This algorithm is applicable whether we are fitting differential equations or algebraic/transcendental equations. FINLIE is also indebted--especially in fitting differential equations--to the following sources:

(a) Theodore R. Goodman of Oceanics, Inc., Plainview, New York, who first called to our attention (in a private communication in 1967) the feasibility of fitting ordinary differential equations--rather than their pseudo-solutions--to observed data. Goodman's technique [see References 8 to 11 in the Bibliography] differs from FINLIE's mainly in the manner in which the influence coefficients are obtained.
(b) Gary T. Chapman and Donn B. Kirk of NASA Ames Research Center, Moffett Field, California, who developed what is now commonly referred to as the "Chapman-Kirk" technique for fitting the aerodynamic equations of motion to free-flight data [see References 12 to 16 in the Bibliography]. When applied to differential equations, FINLIE is essentially a general-purpose Chapman-Kirk program with frills.

(c) Robert H. Whyte, of General Electric, Burlington, Vermont, who for a number of years was apparently indefatigable in applying the Chapman-Kirk technique to a variety of problems. References 17 to 31 in the Bibliography are a sampling of Whyte's reports on his labors in this field.* Many of the handy features of Whyte's programs (for example, the ability to handle multi-round data and to consider any paramic value as fixed or adjustable) have found their way into FINLIE (where they apply to algebraic/transcendental equations as well). It was through my efforts to adapt one of Whyte's specialized programs to our needs that I decided that what was needed was a more general-purpose Chapman-Kirk program. Thus, the idea for FINLIE was conceived. (Unfortunately, the gestation period exceeded that of an elephant.)

* It should be noted that in applying Chapman-Kirk to the 6D equations of motion, Whyte used an unweighted least squares criterion. Since the angular and translational residuals of the fit are not of equal magnitude, Whyte was forced to decouple the angular equations from the translational equations. Disatisfaction with this enforced and often unrealistic decoupling led Whyte and Hathaway to abandon an unweighted least squares in favor of a weighted maximum likelihood criterion. Since this criterion was derived on the assumption of a normal error distribution, their Maximum Likelihood Method [see References 35-39 in the Bibliography] should yield the same final fit (albeit by a different path) as a comparably weighted least squares approach.
BIBLIOGRAPHY

A. Marquardt's Algorithm


(4) Philip R. Bevington, Data Reduction and Error Analysis for the Physical Sciences, McGraw-Hill, New York, 1969. [Lists and discusses a FORTRAN subroutine CURFIT for fitting a single nonlinear equation of the type $y=g(x,P)$ with the aid of Marquardt's algorithm.]


(6) James W. Bradley, "Application of Marquardt's Nonlinear Least Squares Algorithm to Free-Flight Yaw Data Analysis," Ballistic Research Laboratories Memorandum Report No. 2526, September 1975. AD A016906. [Lists and discusses a FORTRAN subroutine MARQ, an offspring of the subroutine CURFIT given in (4) above. The minor refinements in Marquardt's algorithm that have been written into FINLIE are discussed in detail here.]

(7) Keyboard 1978/3 (a publication of Hewlett-Packard Desktop Computer Division). This issue mentions the availability of a "9845 Nonlinear Regression Software" package (09845-15040) for use with the HP System 45. I'm not familiar with the program, but to quote, "This software pack contains programs using Marquardt's Method to fit nonlinear models using up to ten parameters."
BIBLIOGRAPHY (continued)

E. Goodman's Method


C. The Chapman-Kirk Technique


BIBLIOGRAPHY (continued)

D. Whyte's Applications of the Chapman-Kirk Technique

("RHW" in the following denotes Robert H. Whyte; "GE" denotes General Electric Company, Armament Systems Department, Burlington, VT 05401.)


(26) RHW, Angela Jeung and James W. Bradley, "Chapman-Kirk Reduction of Free-Flight Range Data to Obtain Nonlinear Aerodynamic Coefficients," Ballistic Research Laboratories Memorandum Report No. 2298, May 1973. AD 762148. [Covers much the same material as (21) above, but from a different viewpoint.]
BIBLIOGRAPHY (continued)


E. Miscellaneous

(32) **Monte W. Coleman, "MERSION Integration Routine," SPB-10-70, August 1970. [One of a series of informal bulletins issued by what was then the Computer Support Division and is now the Management Information Systems Support Division (MISSD) of BRL.]**

BIBLIOGRAPHY (continued)


F. Whyte and Hathaway's Maximum Likelihood Method

("RHW" and "WHH" in the following denote Robert H. Whyte and Wayne H. Hathaway.)


LIST OF SYMBOLS

\[ A(P) = \sum_{n=1}^{NR} a_{En}, \text{ an N by N multi-round matrix.} \]

\[ a_{jk} \]
the \((j,k)\)-th element of matrix \(A\). \([j,k=1,2,...N]\)

\[ \tilde{a}_{jk} \]
\((a_{jj}a_{kk})^{-\frac{1}{2}} a_{jk} \) [nondimensional]

\[ \beta_{k} \]
\[ b_{k} \]
the \(k\)-th component of \(\vec{b}\).

\[ b_{k} \]
\[(a_{kk})^{-\frac{1}{2}} b_{k} \] [nondimensional]

\( C \)
the vector of \(N3\) parameters.

\( \text{COND} \)
a ROMA input argument vector of \(N2\) single-round initial conditions.

\( \text{CR} \)
\[ c(r_{n})/\epsilon(P_{n-1}), \text{ FINLIE's measure of convergence.} \]

\( c_{j} \)
the \(j\)-th parameter. \([j=1,2,...N3]\)

\( D_{jk}(x_{m},Q) \)
\[ \partial y_{j}(x_{m},Q)/\partial q_{k}, \text{ the } (j,k)-\text{th influence coefficient} \]
evaluated at \(x_{m}\), using the current point \(Q\). \([j=1,2,...N2; \ k=1,2,...N23]\)

\( \text{DU} \)
a ROME output argument vector, Eq. (74).

\( \text{EK} \)
a DUBLIN output argument vector of crude estimates \(s_{k}\). \([k=1,2,...N]\)

\( En \)
the \(n\)-th round identifier. \([n=1,2,...NR]\)

\( \text{EPS} \)
a DUBLIN output argument: \(\epsilon(P)\), the value of \(\epsilon\) at the point currently stored in argument \(P\).

\( F \)
a DUBLIN input argument vector of \(N\) adjust-or-hold-fixed flags associated with the \(N\) params \(p_{k}\).

\( \text{FLAG} \)
a COMMON/NAPILES/ input vector to ROME (and ROMA) containing the \(N23\) single-round adjust-or-hold-fixed flags associated with the \(N23\) params \(q_{k}\).
LIST OF SYMBOLS (continued)

\( f_j \)  
\( y_j', \) system (1)

\( g_j \)  
\( y_j, \) system (2)

h  
a nondimensional positive constant in the steepest descent technique, Eq. (55).

IC  
the set of multi-round initial conditions.

N  
\((NR \times N2) + N3\), the number of parameters in the system.

NA  
\(N2 \times N23\), the number of influence equations for system (1).

NB  
\(N1 \times N23\), the number of needed influence equations for system (2).

NC  
a DUBLIN I/O argument: originally zero (the "first-call" flag), it becomes the number of parameters adjusted.

NF  
a DUBLIN input argument: 1 to fit system (1); 0 to fit system (2).

NM  
a DUBLIN input argument vector, where NM(J) is the number of data points \( R_m \) in the j-th round.

NR  
a DUBLIN input argument: the number of rounds (hence the number of distinct sets of initial conditions to be determined).

NS  
a DUBLIN output argument: 0 means "CALL again"; 1 means "a disaster has occurred"; 2 means "convergence by FINLiE's criterion".

NW  
a DUBLIN input argument: 0 to use the user's weights; 1 to set all weights at unity.

N1  
a DUBLIN input argument: the number of measured dependent variables.

N2  
a DUBLIN input argument: the number of dependent variables in the system.

N3  
a DUBLIN input argument: the number of parameters in the system.

N4  
the number of data points \( R_m \) for all the rounds.
LIST OF SYMBOLS (continued)

N5  a ROME input argument: the number of equations (N2 differential equations plus NA influence equations) to be defined in ROME.

N23  N2 + N3, the number of paramics in a single round.

P  (a) a set of N multi-round paramics;
    (b) a point in the N-dimensional paramic space;
    (c) a DUBLIN I/O argument vector containing the N paramic values.

P̂  the vector from the origin to point P in the N-dimensional paramic space.

̂P  the value of point P that minimizes ε.

PAR  a COMMON/NAPLES/ input vector to ROME (and ROMA) containing the N3 parameters

P_n  the value of point P at the end of the n-th iteration. [n=0,1,...]

P_nA  a candidate for point P_n, obtained by setting λ=λ_nA [n=1,2,...]

paramic  parameter or initial condition.

P_k  the k-th paramic (k=1,2,...N) where the order is: first-round initial conditions, second-round initial conditions,... and finally the N3 parameters.

P̂_k  (a_kk) \frac{1}{2} P_k, the k-th nondimensional paramic.

Q  the single-round equivalent of P.

̂Q  the value of point Q that minimizes γ.

q_k  the k-th single-round paramic, where the order is: the N2 initial conditions, then the N3 parameters.

R  a DUBLIN output argument: the N1 by N4 matrix of residuals, Eq. (83), at the current point P.

RL  a DUBLIN I/O argument: initially set at 0.0 to avoid Marquardt's algorithm; 1.0 to invoke it.

R_m  the m-th data point, consisting of x_m and the N1 dependent variable measurements \bar{y}_{jm}.
LIST OF SYMBOLS (continued)

ROMA an arbitrary name for the user's subroutine defining system (2).

ROME an arbitrary name for the user's subroutine defining system (1).

RS a DUBLIN output argument vector of N1 error measures, Eq. (84).

round an experiment at which measurements were taken and with which a distinct set of initial condition values can be associated.

S the N-dimensional space in which point P has coordinates \((p_1, p_2, \ldots, p_N)\).

\(\tilde{S}\) the N-dimensional scaled space in which point P has coordinates \((\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_N)\).

\(S_1\) the N23-dimensional single-round space in which point Q has coordinates \((q_1, q_2, \ldots, q_{N23})\).

SIG a DUBLIN output argument: the value of s at the current point P.

s estimated standard deviation of the fit, Eq. (69).

\(s_k\) the crude estimated standard deviation in paramic \(p_k\), Eqs. (70-71).

U a ROME input argument vector, Eq. (73); a ROMA output argument vector.

u \(\exp[c_1(x-x_0)]\) in system (4).

W a DUBLIN I/C argument: the N1 by N4 matrix of weights \(w_{jm}\).

\(w_{jm}\) the non-negative weighting factor associated with \(\tilde{y}_{jm}\).

X a DUBLIN input argument vector of the N4 values \(x_m\).

XE a ROME and ROMA input argument: the value of x.

XO a DUBLIN and ROMA input argument: \(x_0\).

x the independent variable of the system.
LIST OF SYMBOLS (continued)

$x_m$ the m-th value of $x$ at which measurements were taken.

$x_0$ the value of the independent variable at which all initial conditions apply.

$Y$ (a) a vector of $N_2$ dependent variables; (b) a DUBLIN input argument: the $N_1$ by $N_4$ matrix of dependent variable measurements $Y_{jm}$.

$Y_C$ a DUBLIN output argument: the $N_2$ by $N_4$ matrix of computed dependent variable values $y_j(x_m, P)$.

$Y_0$ a vector of $N_2$ single-round initial conditions.

$y_j$ the j-th dependent variable. $[j=1,2,...N_2]$

$y_{jm}$ the measured value of $y_j$ at $x_m$ $[j=1,2,...N_1; m=1,2,...N_4]$

$y_j(x_m, P)$ the calculated value of $y_j$ at $x_m$, using the current point $P$.

$\alpha(Q)$ the $N_3$ by $N_3$ single-round matrix at point $Q$

$\alpha_{En}$ the $N$ by $N$ expansion of the matrix $\alpha$ associated with round $En$.

$\alpha_{kn}(Q)$ the $(k,n)$-th element of $\alpha(Q)$, Eqs. (36-38).

$\vec{\beta}(Q)$ a vector point function of $Q$: the vector in space $S_1$ in the direction of the negative gradient of $\gamma$ at point $Q$.

$\vec{\beta}_{En}$ the $N$-dimensional expansion of the vector $\vec{\beta}$ associated with round $En$.

$\beta_k$ the $k$-th component of $\vec{\beta}$. $[k=1,2,...N_3]$

$\Delta P_n$ the increment vector in $S$ from point $P_n$ to the nearby point $P_{n+1}$.

$\Delta Q_n$ the increment vector in $S_1$ from point $Q_n$ to the nearby point $Q_{n+1}$.

$\gamma(Q)$ the single-round equivalent of $\epsilon(P)$, Eq. (14).
LIST OF SYMBOLS (continued)

\( \varepsilon(P) \)  
the nondimensional sum of the weighted squares of the residuals, Eq. (7); the value of the scalar point function \( \varepsilon \) at point \( P \).

\( \lambda \)  
a nonnegative constant added to each diagonal element of the scaled form of matrix \( A \) and adjusted by Marquardt's algorithm so that \( \varepsilon(P_{n+1}) < \varepsilon(P_n) \).

\( \lambda_n^A, \lambda_n^B \)  
two consecutive trial values assigned to \( \lambda \) in an effort to move from point \( P_n \), where \( \lambda_n^B = 10 \lambda_n^A \).

\textbf{Superscripts}  

\(^{(*)}\)  
a row vector

\(^{(*)}T\)  
the transpose of a row vector: that is, a column vector

\(^{(\dagger)}\)  
the scaled (hence nondimensional) form of ( ).

\( ( \ )' \)  
\( d(\ )/dx \)

\textbf{Subscripts}  

\( [\ ]_d \)  
denotes the dimensions of [ ]

\( ( \ )_S \)  
the components of ( ) are in space \( S \)

\( ( \ )_{\tilde{S}} \)  
the components of ( ) are in space \( \tilde{S} \)

\( ( \ )_{S_1} \)  
the components of ( ) are in space \( S_1 \).
APPENDIX

**** FINLIE ****

SUBROUTINE DUBLIN (ROME, NF, N1, N2, N3, N7, N8, NR, NM, XO, X, Y, F, NW, W, P, NL, NC, YC, RS, EPS, SIG, EK, NS)

INPUT ARGUMENTS ...

ROME = THE DUMMY NAME OF THE SUBROUTINE (WRITTEN BY THE USER) THAT DEFINES EITHER
(A) THE DEPENDENT VARIABLES (NF = 0)
OR (B) THE DERIVATIVES OF THE DEPENDENT VARIABLES
(NF .NE. 0)

NF = THE FLAG THAT INDICATES THE NATURE OF SUBROUTINE
ROME (SEE PREVIOUS ARGUMENT)

N1 = NUMBER OF MEASURED DEPENDENT VARIABLES
(1 .LE. N1 .LE. 10)

N2 = TOTAL NUMBER OF DEPENDENT VARIABLES
(N1 .LE. N2 .LE. 20)

N3 = THE MAXIMUM NUMBER OF PARAMETERS (NOT COUNTING
INITIAL CONDITIONS) WHOSE VALUES CAN BE DETERMINED
FROM THE USER'S SUBROUTINE ROME (0 .LE. N3 .LE. 40)

N7 = THE NUMBER OF ROWS IN ARRAYS Y, W AND R BELOW, AS
DIMENSIONED IN THE CALLING PROGRAM (N7 .GE. N1)

N8 = THE NUMBER OF ROWS IN ARRAY YC BELOW, AS
DIMENSIONED IN THE CALLING PROGRAM (N8 .GE. N2)

NR = THE NUMBER OF ROUNDS (INDIVIDUAL CASES) TO BE REDUCED
SIMULTANEOUSLY (1 .LE. NR .LE. (60 - N3)/N2)

NM = A VECTOR OF NR ELEMENTS, WHERE
NM(J) = THE NUMBER OF INDEPENDENT VARIABLE VALUES AT
WHICH MEASUREMENTS WERE TAKEN FOR THE J-TH
ROUND

NOTE ... WE DEFINE
N4 = NM(1) + NM(2) + ... + NM(NR)
= THE TOTAL NUMBER OF INDEPENDENT VARIABLE
VALUES FOR ALL THE ROUNDS

N = NR*N2 + N3
= THE NO. OF ELEMENTS IN P BELOW

NMAX = THE MAXIMUM ELEMENT OF ARRAY NM

THEN WE MUST HAVE
N .LT. N4 .LE. 1000
N1*NMAX .LE. 200
N2*NMAX .LE. 400

XO = THE REFERENCE INDEPENDENT VARIABLE VALUE AT WHICH
ALL INITIAL CONDITIONS APPLY. NOTE ... INPUTS XO,
N1, N2 AND N3 ARE ASSUMED TO HAVE THE SAME VALUE FOR
EACH ROUND.

X = A VECTOR OF THE N4 INDEPENDENT VARIABLE VALUES AT
WHICH MEASUREMENTS WERE TAKEN, WHERE
X(1), ..., X(NM(1)) FOR THE FIRST ROUND
X(NM(1)+1), ..., X(NM(1)+NM(2)) FOR THE SECOND ROUND
ETC.

Y = THE N1 BY N4 MATRIX OF MEASURED DEPENDENT VARIABLE
VALUES, WHERE
Y(I,J) = THE MEASURED VALUE OF THE I-TH DEPENDENT
VARIABLE AT X(J)

F = THE VECTOR OF N FLAGS FOR ARGUMENT P BELOW, WHERE
F(J) = 0.0 IF THE VALUE OF P(J) IS FIXED
F(J) = 1.0 IF THE CURRENT VALUE OF P(J) IS TO BE
ADJUSTED BY THE FITTING PROCESS
NW = THE WEIGHT FLAG ASSOCIATED WITH W BELOW. WHERE
NW = 0 IF THE USER'S WEIGHTS IN ARGUMENT W ARE TO
BE USED
NW = 1 IF ALL WEIGHTS ARE UNITY. IN THIS EVENT, THE
FIRST TIME THIS SUBROUTINE IS CALLED, ALL
ELEMENTS OF MATRIX W ARE SET TO 1.0. (HENCE, THE
USER NEED NOT ESTABLISH W WHEN ALL ITS
ELEMENTS ARE 1.0.)
W = THE N1 BY N4 MATRIX OF WEIGHTS ASSOCIATED WITH INPUT
Y ABOVE. SEE ARGUMENT NW ABOVE.
INPUT/OUTPUT ARGUMENTS...
P = THE CURRENT POINT AT WHICH OTHER ARGUMENTS ARE
EVALUATED. WHERE
P(1)........P(N2) = I.C. FOR FIRST ROUND
P(N2+1)........P(2*N2) = I.C. FOR SECOND ROUND
....
P(NR*N2-N2+1)........P(NR*N2) = I.C. FOR LAST ROUND
P(NR*N2-N2)........P(N) = PARAMETERS
RL = A MARQUARDT ARGUMENT. BEFORE DUBLIN IS CALLED THE
FIRST TIME:
SET RL = 0.0 IF THE MARQUARDT ALGORITHM IS TO BE
OMITTED. THEREAFTER, RL WILL REMAIN
AT 0.0.
SET RL = 1.0 IF THE MARQUARDT ALGORITHM IS TO BE
USED. THEREAFTER, RL UPON RETURN WILL
BE MARQUARDT'S LAMBDA. (HENCE, USE A
NAME, NOT 1.0, IN THE CALL LIST.)
NC = THE 'FIRST CALL' FLAG. BEFORE THIS SUBROUTINE IS
CALLED FOR THE FIRST TIME, NC MUST BE SET TO 0.
THIS SUBROUTINE THEN ESTABLISHES NC AS THE ACTUAL
NUMBER OF INITIAL CONDITIONS AND PARAMETERS BEING
DETERMINED (1.I.E. NC LE. N)
OUTPUT ARGUMENTS...
YC = THE N2 BY N4 MATRIX OF COMPUTED DEPENDENT VARIABLE
VALUES AT THE POINT CONCURRENTLY STORED IN ARRAY P,
WHERE YC(J+K) = COMPUTED VALUE OF THE J-TH DEPENDENT
VARIABLE AT X(K)
R = THE N1 BY N4 MATRIX OF RESIDUALS, WHERE
R(I,J) = Y(I,J) - YC(I,J)
RS = THE VECTOR OF N1 ERROR MEASURES, WHERE
RS(I) = WEIGHTED SUM OF THE SQUARES OF THE
RESIDUALS IN THE I-TH MEASURED DEPENDENT
VARIABLE
EPS = THE ERROR MEASURE OF THE FIT AT THE POINT
CONCURRENTLY STORED IN ARRAY P. EPS IS THE WEIGHTED
SUM OF THE SQUARES OF THE RESIDUALS OVER ALL THE
POINTS, OVER ALL THE MEASURED DEPENDENT VARIABLES
AND OVER ALL THE ROUNDS.
SIG = THE ESTIMATED STANDARD DEVIATION OF THE FIT
EK = THE VECTOR OF CRUDELY ESTIMATED STANDARD DEVIATIONS
IN THE ELEMENTS OF POINT P.
NS = OUTPUT FLAG, WHERE
NS = 0 IF THE PROCESS HAS NOT YET CONVERGED BY
THE CRITERION BUILT INTO SUBROUTINE DUBLIN
= 1 IF ALL OUTPUT ARGUMENTS ARE INVALID (PROBABLY
BECAUSE SOME INPUT ARGUMENTS ARE TOO LARGE
FOR CERTAIN DIMENSIONED ARRAYS). THE CALLING
PROGRAM SHOULD TAKE SPECIAL ACTION (E.G.)
* STOP) IN THIS EVENT.
* = 2 IF THE PROCESS HAS SATISFIED THE BUILT-IN
  CONVERGENCE CRITERION

**

DIMENSION NM(N),X(1),Y(N7,1),F(1),W(N7,1),P(1),Y(1),R(N7,1),
1 RS(1),EK(1)
DIMENSION ALFA(3600),ALF(60,60),GAMMA(60,60),BATA(60),S(60),
1 PA(60),PB(60),PC(60),ALPHA(60,60),BETA(60)
EXTERNAL ROME

*** PART 1. PRELIMINARIES

M1 = N1
M2 = N2
M3 = N3
M7 = N7
M8 = N8
MC = NC
MR = NR
QL = RL
EA = EPS
IF (MC > GT, 0) GOTO 40

THE FIRST TIME DUBLIN IS CALLED (INPUT NC = 0),
SET ALL WEIGHTS TO 1.0 IF NW = 1, THEN EVALUATE
ALPHA, BETA, YC, R, RS, EPS AND SIG AT INPUT
POINT P. OUTPUT NC IS THE NUMBER OF PARAMETERS AND
I.C. TO BE DETERMINED. IF THE MARQUARDT ALGORITHM
IS TO BE USED, SET RL = .01. RETURN.

M4 = 0
DO 5 J = 1, MR
  M4 = M4 + NM(J)
5 CONTINUE
IF (NW < NE, 1) GOTO 30
DO 20 J = 1, M4
  DO 10 I = 1, M1
    W(I,J) = 1.0
20 CONTINUE
10 CONTINUE
30 CALL LONDON (ROME, N1, M1, M2, M3, M7, M8, 60, MR, NM, X0, X, Y, W, F, P,
  1 EA, MC, ALPHA, BETA, YC, R, RS, NS)
EPS = FA
EM = M4 - MC
IF (EM < GT, 0) SIG = SQRT(EPS/EM)
IF (EM < LT, 0) SIG = 0.
NC = MC
IF (QL < NE, 0) RL = 0.01
RETURN

*** PART 2. ON SUBSEQUENT DUBLIN CALLS, DECREASE THE INPUT RL.
SHRINK INPUT ALPHA, BETA AND P TO ALF, BATA AND PA
BY ELIMINATING ALL 'FIXED' (F(K) = 0.0) COMPONENTS.

40 CONTINUE
IF (QL < GT, 0.5E-16) QL = 0.1*QL
LD = MR*MP + 43
JA = 0
JA = 0
DO 60 J = 1, LD
   IF (F(J) .EQ. 0.) GOTO 60
DO 50 K = 1, LD
   IF (F(K) .EQ. J) GOTO 50
   JB = JB + 1
   ALFA(JB) = ALPHAK(J, K)
50 CONTINUE
   JA = JA + 1
   DATA(JA) = RETA(J)
   PA(JA) = P(J)

60 CONTINUE
   JB = 0
DO 80 J = 1, MC
   DO 70 K = 1, MC
      JR = JB + 1
      ALF(K, J) = ALFA(JB)
   70 CONTINUE
   CREATE SCALE FACTORS S(J).
   REPLACE BATA WITH SCALED DATA.
   FORM SCALED ALF(J, K) AND STORE ABOVE THE PRINCIPAL
   DIAGONAL AS ALF(K, J).
   FORM BM = THE SQUARE OF THE MAGNITUDE OF THE
   SCALED DATA VECTOR.

   RM = 0.
DO 100 J = 1, MC
   S(J) = 1./SORT(ALF(J, J))
   BATA(J) = S(J)*BATA(J)
   BM = RM + BATA(J)**2
   K = J - 1
90 IF (K .EQ. 0) GOTO 100
   ALF(K, J) = S(J)*S(K)*ALF(J, K)
   K = K - 1
GOTO 90
100 CONTINUE
   FORM MATRIX GAMMA BASED ON THE CURRENT VALUE OF
   MARQUARDT'S LAMBDA.

   DIAG = 1. + QL
110 CONTINUE
   DO 130 J = 1, MC
      GAMMA(J, J) = DIAG
      K = J - 1
120 IF (K .EQ. 0) GOTO 130
      GAMMA(J, K) = ALF(K, J)
      GAMMA(K, J) = ALF(K, J)
      K = K - 1
GOTO 120
130 CONTINUE
   REPLACE GAMMA BY ITS INVERSE.
   CALL MATINV (GAMMA, MC, PR, 60, 0, DOT)

   FORM THE COMPONENTS DOT OF THE SCALED DELTA P VECTOR.
   FORM THE NEW POINT PC.
   FORM DOT = THE DOT PRODUCT OF THE SCALED DATA AND
   THE SCALED DELTA P VECTORS.
C FORM DPM = THE SQUARE OF THE MAGNITUDE OF THE
SCALED DELTA P VECTOR.  
FORM TR = THE SQUARE OF THE COSINE OF THE ANGLE 
BETWEEN BATA AND DELTA P.

C DOT = 0.
DPM = 0.
DO 150 J = 1+MC
  DO 140 K = 1+MC
    DP = DP + BATA(K)*GAMMA(J,K)
  CONTINUE
PC(J) = PA(J) + DP*S(J)
DOT = DOT + DP*BATA(J)
DPM = DPM + DP*DP
150 CONTINUE
TR = DOT*2/(DPM*BM)

C EXPAND PC TO FULL SIZE AS PB, THE CANDIDATE 
REPLACEMENT FOR INPUT POINT P.

K = 1
DO 170 J = 1+LD
  IF (F(J) .EQ. 0.) GOTO 160
  PR(J) = PC(K)
  K = K + 1
GOTO 170
160 PR(J) = P(J)
170 CONTINUE

*** PART 4.  FOR THE CANDIDATE POINT P, OBTAIN THE ERROR MFAASURE 
(AND ASSOCIATED ARRAYS YC, R, RS, ALPHA AND BETA).

180 CONTINUE
CALL LONDON (ROMENFM,M2,M3,M7,M8,MRNM,X0,X1,X2,Y1,Y2,F,PB,
        EBM0,ALPHA,BETA,YC,R,RS,NS)
IF (NS .EQ. 1) RETURN

IF (ER .LE. EA) GOTO 210
IF (QL .EQ. 0.) GOTO 210

IF (TR .GE. .5) GOTO 200
DO 190 J = 1+LD
  PR(J) = P(J) + 0.1*(PR(J)-P(J))
190 CONTINUE
GOTO 180

INCREASE MARQUARDT'S LAMBDA AND GO BACK TO PART 3.

200 CONTINUE
QL = 10.*QL
GOTO 110
PART 5. THE MARQUARDT ITERATIVE PROCESS HAS BEEN COMPLETED SATISFACTOIRLY. UPDATE ERROR MEASURES EPS AND SIG.
TEST FOR CONVERGENCE, UPDATE POINT P AND COMPUTE ERROR ESTIMATES EK.

210 CONTINUE
RL = QL
EPS = EB
SIG = SQRT(EB/EM)
CR = 1.0 - EB/EA
IF (CR .GE. 0.0 AND CR .LT. 0.000010) NS = 2
K = 1
DO 220 J = 1,LD
P(J) = PB(J)
IF (F(J) .EQ. 0.0) GOTO 215
EK(J) = SIG*SQRT(GAMMA(K,K)*DIAG)
K = K + 1
GOTO 220
215 EK(J) = 0.0
220 CONTINUE
RETURN
END

---

SUBROUTINE LONDON (ROMA,NA,N1,N2,N3,N7,NB,NL,NR,NM,XA,X,Y,W,F,P,
1 EPS,NC,ALPHA,BETA,YC,R,RS,I,S)

* FOR A GIVEN SET OF PARAMETER AND I.C. VALUES AND A GIVEN
* MULTI-ROUND SET OF MEASUREMENTS, THIS SUBROUTINE PRODUCES
* THE ERROR MEASURE E-S, THE COMPUTED DEPENDENT VARIABLE
* VALUES, THE RESIDUALS AND THE ALPHA AND BETA ARRAYS FOR THE
* MULTI-ROUND DATA. ALL ARGUMENTS ARE DEFINED IN THE COMMENTS
* WITHIN SUBROUTINE DUBLIN.

DIMENSION NM(1),P(1),F(I),X(I),Y(N7,1),W(N7,1),
1 YC(NB,1),R(N7,1),RS(1),ALPHA(NL,1),BETA(1)
DIMENSION RSQ(10)
C * THE ABOVE DIMENSION ASSUMES THAT N1 .LE. 10
C DIMENSION C(20),CF(20)
C * THE ABOVE DIMENSIONS ASSUME THAT N2 .LE. 20
C DIMENSION CP(40),FP(40)
C * THE ABOVE DIMENSIONS ASSUME THAT N3 .LE. 40
C DIMENSION ALFA(3600),BATA(60)
C * THE ABOVE DIMENSIONS ASSUME THAT N2 + N3 .LE. 60
C DIMENSION RE(200),WR(200),YM(200)
C * THE ABOVE DIMENSIONS ASSUME THAT N1*(MAX. ELEMENT OF NM) .LE. 200
C DIMENSION YCOMP(400)
C * THE ABOVE DIMENSION ASSUMES THAT N2*(MAX. ELEMENT OF NM) .LE. 400
C DIMENSION XX(1000)
C * THE ABOVE DIMENSION ASSUMES THAT NM(1) + NM(2) + ... + NM(NR) .LE. 1000
C EXTERNAL ROMA
C *** PART 1. *** PRELIMINARIES
IS = 0
MR = NR
M1 = N1
M2 = N2
M3 = N3
M23 = M2 + M3
M6 = M2*MR
LC = 1 + M6
LD = M3 + M5
JX = 1-(1+M23)*(LC-M2)
DO 220 N = 1+LD
   ALPHA(N+N) = 0,
   BETA(N) = 0,
   IF (N .EQ. LD) GOTo 220
   MA = N + 1
   DO 210 K = MA+LD
   ALPHA(K+N) = 0,
   ALPHA(N+K) = 0.
210  CONTINUE
220  CONTINUE
   DO 230 I = 1,M1
      RS(I) = 0.
230  CONTINUE
   MC = 0
   DO 240 K = 1,M3
      KA = K + M6
      CP(K) = P(KA)
      FP(K) = F(KA)
      IF (FP(K) .NE. 0.) MC = MC + 1
240  CONTINUE
   JA = 0
   JB = 0
   EP = 0.

C *** PART 2. *** THE DO-LOOP FOR HANDLING MULTIPLE ROUNDS
C
DO 370 JR = 1,MR
   M4 = N4(JR)
   DO 260 M = 1,M4
      IA = (M-I)*M1
      JA = JA + 1
      XX(M) = X(JA)
      DO 250 I = 1,M1
         IM = IA + I
         YM(IM) = Y(I+JA)
         WR(IM) = W(I+JA)
250  CONTINUE
260  CONTINUE
   LA = (JR-1)*M2
   DO 270 K = 1,M2
      LA = LA + 1
      CF(K) = P(LA)
      CF(K) = F(LA)
      IF (CF(K) .NE. 0.) MC = MC + 1
270  CONTINUE
   CALL PARIS (RONE,N1,1,M1,M2,M23,M6,C,CF,CP,FP,WR,XA,XX,YM,
   YCOMP,RE,RSQ,ALFA,BATA,IR)
   IF (IR .NE. 0) GOTo 280
   IS = 1
   PRINT 275

83
275 FORMAT(1H,10X,'UNSUCCESSFUL RETURN FROM SUBROUTINE PARIS.
1/1H,10X,'ALL SUBROUTINE DUBLIN OUTPUTS INVALID.')
RETURN
280 CONTINUE
DO 290 I = 1*M1
   RS(I) = RS(I) + RSQ(I)
   EP = EP - RSQ(I)
290 CONTINUE
DO 310 M = 1*M4
   NI = (M-1)*M1
   NJ = (M-1)*M2
   JB = JB + 1
   DO 300 J = 1*M2
      IM = NI + J
      JM = NJ + J
      YC(J,JB) = YCOMP(JM)
      IF (J .LE. M1) R(J,JB) = RE(IM)
300 CONTINUE
DO 340 N = LA*LB
   K = N
   J = J + K - LA
   ALPHA(N+K) = ALFA(J)
   IF (K .GT. N) ALPHA(K+N) = ALFA(J)
   J = J + 1
   K = K + 1
   IF (K .LE. LB) GOTO 320
320 CONTINUE
   K = LC
   ALPHA(N+K) = ALFA(J)
   ALPHA(K+N) = ALFA(J)
   J = J + 1
   K = K + 1
   IF (K .LE. LD) GOTO 330
330 CONTINUE
   RETA(N) = BATA(JJ)
   JJ = JJ + 1
DO 360 N = LC*LD
   K = N
   J = JX + (1*M23)*N
   ALPHA(N+K) = ALPHA(N+K) + ALFA(J)
   IF (K .GT. N .AND. JR .EQ. MR) ALPHA(K+N) = ALPHA(N+K)
   J = J + 1
   K = K + 1
   IF (K .LE. LD) GOTO 350
350 CONTINUE
   RETA(N) = RETA(N) + BATA(JJ)
   JJ = JJ + 1
360 CONTINUE
370 CONTINUE
EPS = EP
NC = MC
RETURN
END
SUBROUTINE PARIS(ROME, NA, NB, N1, N2, N3, N4, CF, P, *F, W, XA, X, YM, 
YC, R, RSQ, ALFA, BATA, IR)

FOR A GIVEN SET OF PARAMETER AND IC ESTIMATES AND THE GIVEN
MEASUREMENTS FOR A SINGLE ROUND, THIS SUBROUTINE PRODUCES
THE COMPUTED DEPENDENT VARIABLE VALUES, THE RESIDUALS AND
IF NB .NE. 0) THE ALPHA AND BETA ARRAYS FOR THE GIVEN ROUND.

INPUT ARGUMENTS **
ROME = THE DUMMY NAME OF THE SUBROUTINE (WRITTEN BY THE
USER) THAT DEFINES EITHER
(A) THE DEPENDENT VARIABLES (NA = 0)
OR (B) THE DERIVATIVES OF THE DEPENDENT VARIABLES
(NA .NE. 0)
NA = THE FLAG THAT INDICATES THE NATURE OF SUBROUTINE
ROME (SEE PREVIOUS ARGUMENT)
NB = 0 IF ARGUMENTS ALFA AND BATA BELOW ARE NOT TO BE
COMPUTED
1 IF ALFA AND BATA ARE TO BE COMPUTED
N1 = NUMBER OF MEASURED DEPENDENT VARIABLES (N1 .GE. 1)
N2 = TOTAL NUMBER OF DEPENDENT VARIABLES (N2 .GE. N1)
N3 = TOTAL NUMBER OF PARAMETERS AND INITIAL CONDITIONS
(N23 .GE. N2)
N4 = NUMBER OF MEASUREMENTS TAKEN ON EACH OF THE N1
MEASURED VARIABLES
C = VECTOR OF THE N2 INITIAL CONDITION ESTIMATES
CF = VECTOR OF THE N2 INITIAL CONDITION FLAGS
P = VECTOR OF THE N3 PARAMETER VALUES
PF = VECTOR OF THE N3 PARAMETER FLAGS
Y = VECTOR OF THE N1 BY N4 WEIGHTS ASSOCIATED WITH INPUT
YM DEFINED BELOW
WA THE REFERENCE X VALUE AT WHICH INITIAL CONDITIONS
APPLY
X = VECTOR OF THE N4 VALUES OF THE INDEPENDENT VARIABLE
AT WHICH MEASUREMENTS WERE TAKEN
YM = VECTOR OF THE N1 BY N4 MEASURED Y VALUES FOR ONE
ROUND, WHERE
\[ \text{YM}(1) = \text{MEASURED VALUE OF } Y(I) \text{ AT } X(M) \]
\[ I = 1, 2, \ldots, N1 \]
\[ M = 1, 2, \ldots, N4 \]

OUTPUT ARGUMENTS **
YC = VECTOR OF N2 BY N4 COMPUTED Y VALUES FOR ONE ROUND,
WHERE
\[ \text{YC}(1) = \text{COMPUTED VALUE OF } Y(J) \text{ AT } X(M) \]
\[ J = 1, 2, \ldots, N2 \]
R = VECTOR OF N1 BY N4 RESIDUALS, WHERE
\[ \text{R}(1) = \text{YM}(1) - \text{YC}(1) \]
\[ I = 1, 2, \ldots, N1 \]
\[ M = 1, 2, \ldots, N4 \]
RSQ = VECTOR OF N1 ERROR MEASURES, WHERE
\[ \text{RSQ}(1) = \text{WEIGHTED SUM OF SQUARES OF THE RESIDUALS \ IN } \text{THE } I-TH \text{ DEPENDENT VARIABLE} \]
ALFA = VECTOR OF N23 BY N23 ALPHA VALUES, WHERE
\[ \text{ALFA}(1) = \text{ALPHA}(N,K) \]
DIMENSION C(1), CF1), P(1), PF1), X(1), YM1), YC1), R(1), RSQ(1)
DIMENSION U(400), DU(400), S(1000)

DIMENSION C(1), C(1), P(1), PF(1), X(1), Y(1), YM(1), YM(1), R(1), RSQ(1)

COMMON /NAPLES/ PAR(40), FLAG(60)

EXTERNAL ROME
DATA HMIN, DELX, -1.0, .00010, .125/

PART (1) PRELIMINARIES

IR = 0
M21 = N23
M1 = N1
M2 = N2
M4 = N4
M5 = M2(1+M23)
IF (M5 .eq. 0) 5070 24
IR = 1
PRINT 10
FORMAT(IHO,* SUBROUTINE PARIS WARNING ... *)
PRINT 22,M5
FORMAT(IHO,* INCREASE DIMENSIONS OF U AND DU (AND IN SUBROUTINE PARIS)
10 MERO: INCREASE DIMENSIONS OF T, G AND S TO 'I5)
2* M6 = M4*M5
IF (M6 .LE. 10000) GOTO 28
IR = 1
PRINT 10
PRINT 26,M5
FORMAT(IHO,* INCREASE DIMENSIONS OF S TO 'I5)
28 CONTINUE
IF (IR .EQ. 1) RETURN
IM = 0
H = DEL*X(X(M4)-X(1))/FLOAT(M4-1)
JA = M2 + 1
DO 30 JB = JA,M5
U(JR) = 0.0
DU(JB) = 0.0
30 CONTINUE
DO 32 K = 1,M2
FLAG(K) = CF(K)
32 CONTINUE
M3 = M23 - M2
DO 34 K = 1,M3
PAR(K) = P(K)
KA = K + M2
FLAG(KA) = PF(K)
34 CONTINUE

C *** PART (2) DETERMINE KL = NO. OF POINTS IN X LESS THAN XA AND KR = NO. OF POINTS IN X GREATER THAN XA.
C IF XA COINCIDES WITH A POINT IN X, THEN THE 'COMPUTED' Y VALUES AT THAT POINT ARE THE INITIAL CONDITIONS FROM SUBROUTINE RONN.
C
DO 40 M = 1,M4
IF (X(A-X(M)) .LT. 50.0) GOTO 40
40 CONTINUE
KL = M4
KR = 0
GOTO 80
50 KL = M - 1
KR = M4 - M
IM = 1
CALL BONN(M2,M5,C,U)
LB = KL*M5
DO 60 L = 1,M5
LB = LB + 1
S(LA) = U(L)
60 CONTINUE
GOTO 80
70 KL = M - 1
KR = M4 - KL
C *** PART (3) FOR EACH POINT IN X, SOLVE THE SYSTEM OF EQUATIONS DEFINED IN SUBROUTINE ROME TO OBTAIN COMPUTED Y VALUES. THESE Y VALUES ARE STORED IN S.
C
A0 CONTINUE
IF (KL .EQ. 0) GOTO 90
IA = -1
IH = 0
LK = KL
GOTO 100
90 IA = 1
IR = M4 + 1
LK = IH - KR
100 X1 = XA
IF (IM, EQ, 0) CALL BONN(M2, M5, C, U)
IM = 0
110 X = X(LK)
IF (NA, NE, 0) GOTO 114
CALL ROME(C, X1, X2, U)
GOTO 116
114 CONTINUE
HA = DELX*ARS(X2 - X1)
HH = ABS(H)
H = AMIN1(HA, HR)
CALL MERSO(ROME(M5, X1, X2, U, DU, H, HMIN, Q)
116 CONTINUE
LB = (LK-1)*M5
DO 120 L = 1, M5
LA = LB + 1
S(LB) = U(L)
120 CONTINUE
LK = LK + IA
IF (LK, NE, IH) GOTO 110
IF (IA, EQ, 1) GOTO 130
IF (KR, NE, 0) GOTO 90
C*** PART (4) CONDONV S TO VECTOR YC AND FORM THE RESIDUAL
C VECTOR R.
130 CONTINUE
DO 150 M = 1, M4
NR = (M-1)*M1
NY = (M-1)*M2
LA = (M-1)*M5
DO 140 J = 1, M2
IM = NR + J
JM = NY + J
LR = LA + J
YC(JM) = S(LB)
IF (J, LE, M1, AND, W(IM), NE, 0) R(IM) = YM(IM) - YC(JM)
IF (J, LE, M1, AND, W(IM), EQ, 0) R(IM) = 0.
140 CONTINUE
150 CONTINUE
C*** PART (5) COMPUTE VECTOR RSQ
C DO 170 I = 1, M1
RM = 0.0
DO 160 W = 1, M4
IM = I + (M-1)*M1
RM = RM + W(IM)*R(IM)**2
160 CONTINUE
RSQ(1) = RM
170 CONTINUE
IF (NR, EQ, 0) RETURN
C*** PART (6) COMPUTE VECTOR ALFA
C DO 210 K = 1,M23
   LA = K*M2
   NL = (K-1)*M23
   DO 200 N = K*M23
      LB = N*M2
      NK = NL + N
      ALFA(NK) = 0.0
   DO 190 I = 1,M1
      ALF = 0.0
      LC = LA + I
      LD = LB + I
      DO 180 M = 1,M4
         IM = I + (M-1)*M1
         ALF = ALF + W(IM)*S(LC)*S(LD)
         LC = LC + M5
         LD = LD + M5
      CONTINUE
   ALFA(NK) = ALFA(NK) + ALF
   CONTINUE
190 IF (N .EQ. K) GOTO 200
   KN = K + (N-1)*M23
   ALFA(KN) = ALFA(NK)
200 CONTINUE
210 CONTINUE

C *** PART (7) COMPUTE VECTOR BATA
C DO 240 N = 1,M23
   LA = N*M2
   RATA(N) = 0.0
   DO 230 I = 1,M1
      BAT = 0.0
      LR = LA + I
      DO 220 M = 1,M4
         IM = I + (M-1)*M1
         R = BAT + W(IM)*R(IM)*S(LB)
         LR = LB + M5
      CONTINUE
   BATA(N) = RATA(N) + BAT
230 CONTINUE
240 CONTINUE
RETURN
END

C--------------------------------------------------------
C SUBROUTINE RONN(N2,N5,C+U)
C
C THIS SUBROUTINE (CALLED BY SUBROUTINE PARIS) ASSIGNS INITIAL
C CONDITIONS (THE VALUES AT X = XA) TO VECTOR U. FOR THE
C DEFINITIONS OF THE ARGUMENTS, SEE THE COMMENTS IN SUBROUTINE
C PARIS.
C
DIMENSION C(N2),U(N5)
M2 = N2
M5 = N5
10 CONTINUE
LA = 0
DO 30 JA = 1,M2
LD = LA + M2
DO 20 JB = 1,M2
  IF (JA .EQ. J8) U(LB) = 1.0
  IF (JA .NE. JB) U(LB) = 0.0
20 CONTINUE
30 CONTINUE
IF (LR .GE. M5) GOTO 50
LA = LR + 1
DO 40 LB = LA,M5
  U(LB) = 0.0
40 CONTINUE
50 CONTINUE
RETURN
END

SUBROUTINE MFRS0 (FUNC,N,X,Z,Y,F,H,MIMaxE)
DIMENSION Y(I),F(I) $ DIMENSION T(400),G(400),S(400)
LOGICAL RC,RE,RH,BR,BX $ NT=NS ZT=ZS HMIN=MINS ET=ABS(E)
IF(HMIN .LT.0.0) HMIN=0.01*ABS(H) $ BR=BR'=TRUE.
BC=0.0,LT,ET,AND,ET,LT,1.0 $ E5=ET*5.0
IF((ZT,GT,X,AND,H,LT,0.0),OR,(ZT,LT,X,AND,H,GT,0.0)) HA=H
IF(N=T,LE,400) GOTO 100% PRINT 1,NT% STOP
1 FORMAT(22H RUN ERROR MERSION, N=110)
100 XS=X $ DO 110 J=1,NT $ G(J)=Y(J)
110 CONTINUE
200 H3=H$ Q=T*X+H-ZT $ AE=.TRUE.
  IF((Q,LT,0.0,AND,H,GE,0.0),OR,(Q,GT,0.0,AND,H,LE,0.0)) GO TO 210
  H=ZT-X $ RR=.FALSE.
210 H3=H/3.0 $ DO 510 ISW=1,5 $ CALL FUNC(N,T,X,Y,F) $ DO 450 I=1,NT
  Q=W3*F(I) $ GOTO(301,302,303,304,305,306) ISW
450 CONTINUE
301 T(I)=X% GOTO 400
302 R=0.5*(T(I)) $ GOTO 400
303 S(I)=R=3.0*$ R=0.375*(R-T(I)) $ GOTO 400
304 T(I)=R=3.0*$ R=1.5*(R=S(I)) $ GOTO 400
305 R=0.5*(Q+T(I)) $ Q=ABS(2.0*R-1.5*(Q+T(I)))
400 Y(I)=G(I)+R $ IF(ISW,NE,5) GOTO 450 $ IF(.NOT,BC) GOTO 450 $ R=ES
  IF(ARS(Y(I)),GE,0.001) R=R**ABS(Y(I))
  IF((Q,LT,R),(OR,.NOT.,BX)) GOTO 420 $ BR=TRUE.$ BH=FALSE.$ H=0.5*H
  IF(ARS(H),GE,HMIN) GOTO 410 $ M=IS91H(H)*HMIN $ BX=FALSE.
410 DO 411 J=1,NT $ Y(J)=G(J)
411 CONTINUE $ X=X% GOTO 200
420 IF(Q,GE,0.03125) BE=.FALSE.
450 CONTINUE $ GOTO(501,502,503,504,510,512) ISW
501 X=X+H3% GOTO 510
502 X=X+0.5% H3% GOTO 510
503 X=X+0.5% H3% GOTO 510
504 X=X+0.5% H3% GOTO 510
510 CONTINUE $ IF(.NOT,BC) GOTO 521
  IF(.NOT,(RF,AND,MM,AND,RM)) GOTO 520 $ H=2.0H $ BX=TRUE.
520 RM=TRUE.
521 IF(RR) GOTO 100 $ H=HSS$ RETURN $ END

90
SUBROUTINE MATINV

OBTAINED FROM COMPUTER SUPPORT DIVISION
ABERDEEN RESEARCH AND DEVELOPMENT CENTER

SUBROUTINE MATINV(A,N,C,NMAX,K,DET)

DIMENSION A(NMAX),C(1)

N = N
K = K
IF (I1-KK) 3,1,1
N3 = NN
IF (KK) 2,4,2
ASSIGN 9 TO N5
ASSIGN 13 TO N7
GOTO 5
N3 = KK + NN - 1
ASSIGN 10 TO N5
ASSIGN 14 TO N7
DET = 1.0
DO 15 I = 1,NN
   IF (A(I,I)) 7,6,7
   WRITE(6,17)
   DET = 0.0
   GOTO 16
   T1 = 1.0/A(I,I)
   DET = DET*A(I,I)
   A(I,I) = 1.0
   DO 8 J = 1,N3
      A(I,J) = A(I,J)*T1
   CONTINUE
   GOTO N5, (9,10)
   C(I) = C(I)*T1
   DO 14 J = 1,NN
      IF (I-J) 11,14,11
      T1 = A(I,J)
      A(J,I) = 0.0
      DO 12 L = 1,N3
         A(J,L) = A(J,L) - T1*A(I,L)
      CONTINUE
   GOTO N7, (13,14)
   C(J) = C(J) - T1*C(I)
   CONTINUE
   CONTINUE
   CONTINUE
   RETURN
17 FC MAT (16H SINGULAR MATRIX)
END
<table>
<thead>
<tr>
<th>No. of Copies</th>
<th>Organization</th>
<th>No. of Copies</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Commander</td>
<td>1</td>
<td>Commander</td>
</tr>
<tr>
<td></td>
<td>Defense Technical Info Center</td>
<td>US Army Armament Materiel Readiness Command</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cameron Station</td>
<td>ATTN: DDC-DDA</td>
<td>ATTN: DRSAR-LEP-L, Tech Lib</td>
</tr>
<tr>
<td></td>
<td>Alexandria, VA 22314</td>
<td>Rock Island, IL 61299</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td>1</td>
<td>Director</td>
</tr>
<tr>
<td></td>
<td>US Army Materiel Development and Readiness Command</td>
<td>US Army Armament Research and Development Command</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTN: DRCDMD-ST</td>
<td>Benet Weapons Laboratory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5001 Eisenhower Avenue</td>
<td>ATTN: DRDAR-LCB-TL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Alexandria, VA 22333</td>
<td>Watervliet, NY 12189</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Commander</td>
<td>1</td>
<td>Commander</td>
</tr>
<tr>
<td></td>
<td>US Army Armament Research and Development Command</td>
<td>US Army Aviation Research and Development Command</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTN: DRDAR-TSS (2 cys)</td>
<td>ATTN: DRSAV-E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dover, NJ 07801</td>
<td>P.O. Box 209</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>St. Louis, MO 61366</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Commander</td>
<td>1</td>
<td>Director</td>
</tr>
<tr>
<td></td>
<td>US Army Armament Research and Development Command</td>
<td>US Army Air Mobility Research and Development Laboratory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTN: DRDAR-LCA, Mr. W. R. Benson</td>
<td>Ames Research Center</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DRDAR-LCA-F, Mr. A. Loeb</td>
<td>Moffett Field, CA 94035</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DRDAR-LCA-FA, Mr. E. Friedman</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mr. D. Mertz</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dover, NJ 07801</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Commander</td>
<td>1</td>
<td>Commander</td>
</tr>
<tr>
<td></td>
<td>US Army Armament Research and Development Command</td>
<td>US Army Communications Research and Development Command</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTN: DRDAR-LCN, Mr. F. Saxe</td>
<td>ATTN: DRDCO-PPA-SA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mr. F. Saxe, Mr. F. Scerbo</td>
<td>Fort Monmouth, NJ 07703</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dover, NJ 07801</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td>1</td>
<td>Commander</td>
</tr>
<tr>
<td></td>
<td>US Army Armament Research and Development Command</td>
<td>US Army Electronics Research and Development Command</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTN: DRDAR-LCU, Mr. A. Moss</td>
<td>Technical Support Activity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dover, NJ 07801</td>
<td>ATTN: DFLSID-I</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fort Monmouth, NJ 07703</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td>1</td>
<td>Commander</td>
</tr>
<tr>
<td></td>
<td>US Army Missile Command</td>
<td>US Army Missile Command</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATTN: DRSMI-R</td>
<td>ATTN: DR8MI-R</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Redstone Arsenal, AL 35809</td>
<td></td>
</tr>
<tr>
<td>No. of Copies</td>
<td>Organization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>AFATL (Tech Lib)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>AFFDL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Director</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No. of Copies</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Director</td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
</tr>
<tr>
<td>1</td>
<td>Commander</td>
</tr>
</tbody>
</table>

US Army Missile Command
ATTN: DRSMI-YDL
Redstone Arsenal, AL 35809

US Army Tank Automotive Research and Development Command
ATTN: DRDTA-UL
Warren, MI 48090

US Army TRADOC Systems Analysis Activity
ATTN: ATAA-SL, Tech Lib
White Sands Missile Range NM 88002

US Army Yuma Proving Ground
ATTN: STEYP-TMW
Yuma, AZ 85364

US Army Research Office
ATTN: CRD-AA-EH
F.O. Box 12211
Research Triangle Park NC 27709

AFATL (Tech Lib)
Eglin AFB, Fl 32542

AFFDL
Wright-Patterson AFB, OH 45433

National Aeronautics and Space Administration
Ames Research Center
ATTN: Dr. Gary Chapman
Mr. A. Seiff
Mr. Murray Tobak
Tech Lib
Moffett Field, CA 94035

Jet Propulsion Laboratory
ATTN: Tech Lib
Mr. Peter Joffe
4800 Oak Grove Drive
Pasadena, CA 91103

David W. Taylor Naval Ship Research and Development Center
ATTN: Tech Lib
Bethesda, MD 20084

Naval Research Laboratory
ATTN: Tech Info Div
Washington, D.C. 20375

Naval Surface Weapons Center
ATTN: Dr. Thomas Clare
Dr. W.R. Chadwick
Dr. W.G. Soper
Dr. F. Moore
Dr. T.R. Pepitone
Dahlgren, VA 22448

Naval Surface Weapons Center
ATTN: Code 750, Tech Lib
Silver Spring, MD 20910

Naval Weapons Center
ATTN: Code 233
China Lake, CA 93555

Arnold Research Organization, Inc.
Project Support and Special Studies Section
Aerodynamics Division
Projects Branch
ATTN: Dr. John C. Adams, Jr.
Arnold AFB, TN 37380
<table>
<thead>
<tr>
<th>No. of Copies</th>
<th>Organization</th>
<th>No. of Copies</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Director</td>
<td>1</td>
<td>Oceanics, Inc.</td>
</tr>
<tr>
<td></td>
<td>Sandia Laboratories</td>
<td></td>
<td>ATTN: Dr. Theodore R. Goodman</td>
</tr>
<tr>
<td></td>
<td>ATTN: Division 1331, Mr. H.R. Vaughn</td>
<td></td>
<td>Technical Industrial Park</td>
</tr>
<tr>
<td></td>
<td>Mr. A.E. Hodapp, Jr.</td>
<td></td>
<td>Plainview, NY 11803</td>
</tr>
<tr>
<td></td>
<td>Albuquerque, NM 87115</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Director</td>
<td>1</td>
<td>Schering Corporation</td>
</tr>
<tr>
<td></td>
<td>National Aeronautics and Space Administration</td>
<td></td>
<td>ATTN: M. Miller</td>
</tr>
<tr>
<td></td>
<td>George C. Marshall Space Flight Center</td>
<td></td>
<td>60 Orange Street</td>
</tr>
<tr>
<td></td>
<td>ATTN: MS-I, Library</td>
<td></td>
<td>Bloomfield, NJ 07003</td>
</tr>
<tr>
<td></td>
<td>Huntsville, AL 35812</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Director</td>
<td>1</td>
<td>Systems Technology, Inc.</td>
</tr>
<tr>
<td></td>
<td>National Aeronautics and Space Administration</td>
<td></td>
<td>ATTN: Arlene Muise, Librarian</td>
</tr>
<tr>
<td></td>
<td>Langley Research Center</td>
<td></td>
<td>13766 South Hawthorne Blvd.</td>
</tr>
<tr>
<td></td>
<td>ATTN: MS-185, Tech Lib</td>
<td></td>
<td>Hawthorne, CA 90250</td>
</tr>
<tr>
<td></td>
<td>Dr. Clarence Young</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Langley Station</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hampton, VA 23365</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>The Analytic Sciences Corporation (TASC)</td>
<td></td>
<td>Union Research Center</td>
</tr>
<tr>
<td></td>
<td>ATTN: Mr. James E. Kai: 6 Jacob Way</td>
<td></td>
<td>ATTN: Dr. M. A. Selim</td>
</tr>
<tr>
<td></td>
<td>Reading, MA 01867</td>
<td></td>
<td>P.O. Box 76</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Brea, CA 92621</td>
</tr>
<tr>
<td>1</td>
<td>E.I. du Pont de Nemours and Company</td>
<td></td>
<td>Case Western Reserve</td>
</tr>
<tr>
<td></td>
<td>Engineering Department</td>
<td></td>
<td>ATTN: Professor Philip R Bevington</td>
</tr>
<tr>
<td></td>
<td>ATTN: Dr. Donald W. Marquardt</td>
<td></td>
<td>Cleveland, OH 44106</td>
</tr>
<tr>
<td></td>
<td>Wilmington, DE 19898</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Exxon Research Center</td>
<td></td>
<td>Oklahoma State University</td>
</tr>
<tr>
<td></td>
<td>ATTN: Mr. John Stevens</td>
<td></td>
<td>ATTN: J. P. Chandler</td>
</tr>
<tr>
<td></td>
<td>Building 28, P.O. Box 45</td>
<td></td>
<td>Stillwater, OK 74074</td>
</tr>
<tr>
<td></td>
<td>Linden, NJ 07036</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>General Electric Company</td>
<td></td>
<td>University of Virginia</td>
</tr>
<tr>
<td></td>
<td>Armament Systems Department</td>
<td></td>
<td>ATTN: Professor Dr. Jacobson</td>
</tr>
<tr>
<td></td>
<td>ATTN: Mr. Robert H. Whyte</td>
<td></td>
<td>Thornton Hall</td>
</tr>
<tr>
<td></td>
<td>Lakeside Avenue</td>
<td></td>
<td>Charlottesville, VA 22904</td>
</tr>
<tr>
<td></td>
<td>Burlington, VT 05401</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Aberdeen Proving Ground
Dir. USAMRAA
ATTN: DRASY-P
DRASY-EP, H. Cohen
Dir. USAMCOM
ATTN: DRMP 384
Dir. USACE, Rmp. Eng., E.A.
ATTN: DRMRF 490
USER EVALUATION OF REPORT

Please take a few minutes to answer the questions below; tear out this sheet, fold as indicated, staple or tape closed, and place in the mail. Your comments will provide us with information for improving future reports.

1. BRL Report Number ____________________________________________

2. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which report will be used.) ____________________________________________

3. How, specifically, is the report being used? (Information source, design data or procedure, management procedure, source of ideas, etc.) ____________________________________________

4. Has the information in this report led to any quantitative savings as far as man-hours/contract dollars saved, operating costs avoided, efficiencies achieved, etc.? If so, please elaborate. ____________________________________________

5. General Comments (Indicate what you think should be changed to make this report and future reports of this type more responsive to your needs, more usable, improve readability, etc.) ____________________________________________

6. If you would like to be contacted by the personnel who prepared this report to raise specific questions or discuss the topic, please fill in the following information.

   Name: ____________________________________________

   Telephone Number: ____________________________________________

   Organization Address: ____________________________________________