Technical Memorandum No. 33-203

A Users' Guide for SEARCH and ITER

N. S. Newhall

[illegible]

H. Fred Lesh, III, Manager
Computer Applications

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

April 15, 1965

Prepared Under Contract No. NAS 7-100
National Aeronautics & Space Administration

CONTENTS

TABLES

FIGURES

ABSTRACT

30119

With the creation of the Jet Propulsion Laboratory's new trajectory monitor (JPTRAJ) came a unique ability. Programming options, once a cumbersome part of the old trajectory program, are now separate entities tied together in a JPTRAJ source deck. Two of the programs are described in this Report: SEARCH, a program to calculate initial values of a trajectory (or any computation) whose end conditions are known, and ITER, an iterative program simulating a "DO-loop" in a source deck. Both programs operate on an IBM 7094 under JPTRAJ, running under either the SFOF or FORTRAN system.

I. INTRODUCTION

This Report is to serve as a users' and programmers' guide to two independent programs, SEARCH and ITER. SEARCH arose from requirements of the trajectory and midcourse maneuver programs. Formerly a part of the Jet Propulsion Laboratory's (JPL) trajectory system, it is now a separate program under JPTRAJ (Jet Propulsion Laboratory Trajectory Monitor). Users know *a priori* the end conditions of a trajectory and must compute its initial state. SEARCH may be used with any sequence of programs in a source deck to match end conditions with their initial values.

The purpose of the iterative program ITER is to enable JPTRAJ users to run multiple cases of any program sequence and vary selected parameters between cases. By using ITER properly, a user may, for example, run a parameter study over a program with only a small JPTRAJ source deck.

Both SEARCH and ITER are iterative programs under JPTRAJ. They may operate in Mode 2, Mode 4, or job-shop mode.

II. THEORY IN SEARCH

The iteration principle used in SEARCH is known as the Newton – Raphson method. The simplest illustration of this method is the so-called "one-by-one" (one *dependent* variable and one *independent* variable) case. Suppose we have a function $y = f(x)$ and we are given some desired value y_T. We want to find the solution x_T such that $y_T = f(x_T)$.

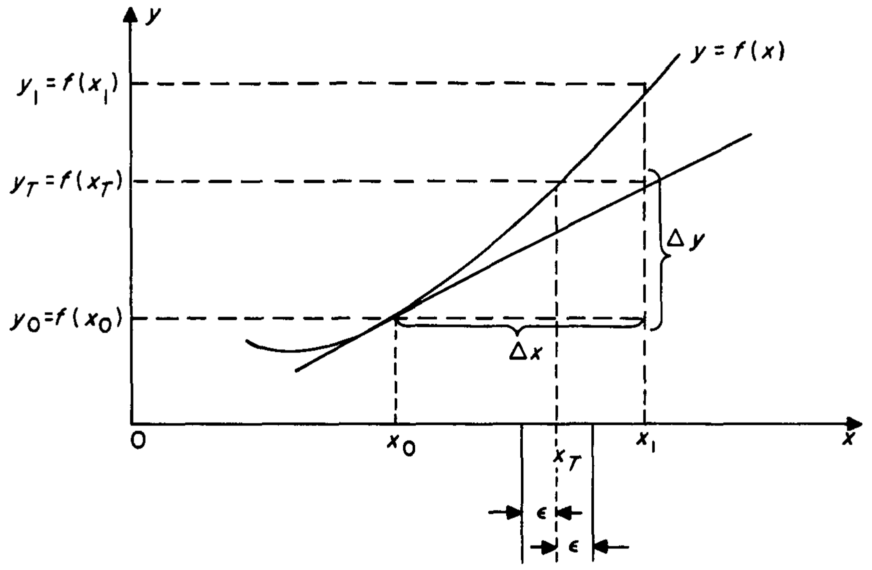


Fig. 1. 1-by-1 Newton-Raphson approximation

We pick (or are given) some value x_0 near the solution x_T (Fig. 1). If $f(x)$ is almost linear in this neighborhood, we can approximate it by the tangent at x_0. Let

$$\Delta y = y_T - y_0$$

(the error in y). From Fig. 1, the y_T on the tangent is at x_1. Let

$$\Delta x = x_1 - x_0.$$

Then clearly

$$f'(x_0) = \frac{\Delta y}{\Delta x}$$

or

$$\Delta x = \frac{\Delta y}{f'(x_0)} = \frac{y_T - y_0}{f'(x_0)}$$

Finally,

$$x_1 = x_0 + \Delta x = x_0 + \frac{\Delta y}{f'(x_0)}$$

In this manner a better approximation to x_T will be obtained. Repeating the process at x_1, we may again improve the result and may, in fact, iterate as many times as necessary to obtain some $x_i = x_T \pm \epsilon$, where ϵ is a permissable error tolerance.

Note that if $y = f(x)$ is not nearly linear in the neighborhood concerned, it will not be approximated by a tangent, and the method loses its value.

In SEARCH, an extension of this principle is used. The general case is the n by n case (n *independent* variables and n *dependent* variables). The function f is a vector function, and x and y are n-dimensional vectors. The derivative f' takes the form of a matrix of partial derivatives and is denoted as **A**. Also, the allowable error ϵ is a vector.

Thus

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

and

$$\Delta \mathbf{x} = \mathbf{x}_1 - \mathbf{x}_0 = \mathbf{A}_0^{-1} \Delta \mathbf{y}$$

and convergence is defined as

$$\left| y_{ik} - y_{Tk} \right| \leq \epsilon_k, \; k = 1, 2, \cdots, n$$

Example: A typical search over a trajectory will be 3 by 3 with $\dot{x}$, $\dot{y}$, and $\dot{z}$ as independent variables and $\mathbf{B \cdot T}$, $\mathbf{B \cdot R}$ and t_f as dependent variables.

Thus,

$$\mathbf{x} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} \mathbf{B \cdot T} \\ \mathbf{B \cdot R} \\ t_f \end{pmatrix}$$

and

$$\mathbf{A}_0 = \begin{bmatrix} \dfrac{\partial \mathbf{B \cdot T}}{\partial \dot{x}_0} & \dfrac{\partial \mathbf{B \cdot T}}{\partial \dot{y}_0} & \dfrac{\partial \mathbf{B \cdot T}}{\partial \dot{z}_0} \\ \dfrac{\partial \mathbf{B \cdot R}}{\partial \dot{x}_0} & \dfrac{\partial \mathbf{B \cdot R}}{\partial \dot{y}_0} & \dfrac{\partial \mathbf{B \cdot R}}{\partial \dot{z}_0} \\ \dfrac{\partial t_f}{\partial \dot{x}_0} & \dfrac{\partial t_f}{\partial \dot{y}_0} & \dfrac{\partial t_f}{\partial \dot{z}_0} \end{bmatrix}$$

In SEARCH itself there are a few differences from a purely analytical approach.

1. The matrix $\mathbf{A}_0 = \mathbf{f}'(\mathbf{x}_0)$ may not be immediately available and must be obtained by numerical differencing. By input option, SEARCH either will accept the first $\mathbf{A}_0$ matrix from another program, such as a conic, or will compute it. If no external partials are available, a nominal case is run using given initial conditions ($\mathbf{x}$). Then n perturbed cases are run, each case representing a different perturbed initial condition. (The amount of perturbation δ_i of each initial condition must be input.)

   The results from all these runs are then differenced to yield

$$\frac{\partial y_i}{\partial x_j} \approx \frac{y_{ip} - y_{in}}{x_{jp} - x_{jn}} = a_{ij}$$

where p = perturbed, n = nominal, and the a_{ij}'s are the elements of the $\mathbf{A}_0$ matrix.

2. Because the procedure for obtaining the $\mathbf{A}_0$ matrix requires so much calculation, a new one is not necessarily formed at the beginning of each iteration. One may employ the same set of partials for any number of iterations. (The number usually used is 3.)

3. If, after a certain number of iterations, the computed results do not lie within the specified tolerance of the target values, the search is termed divergent and an error exit is taken.

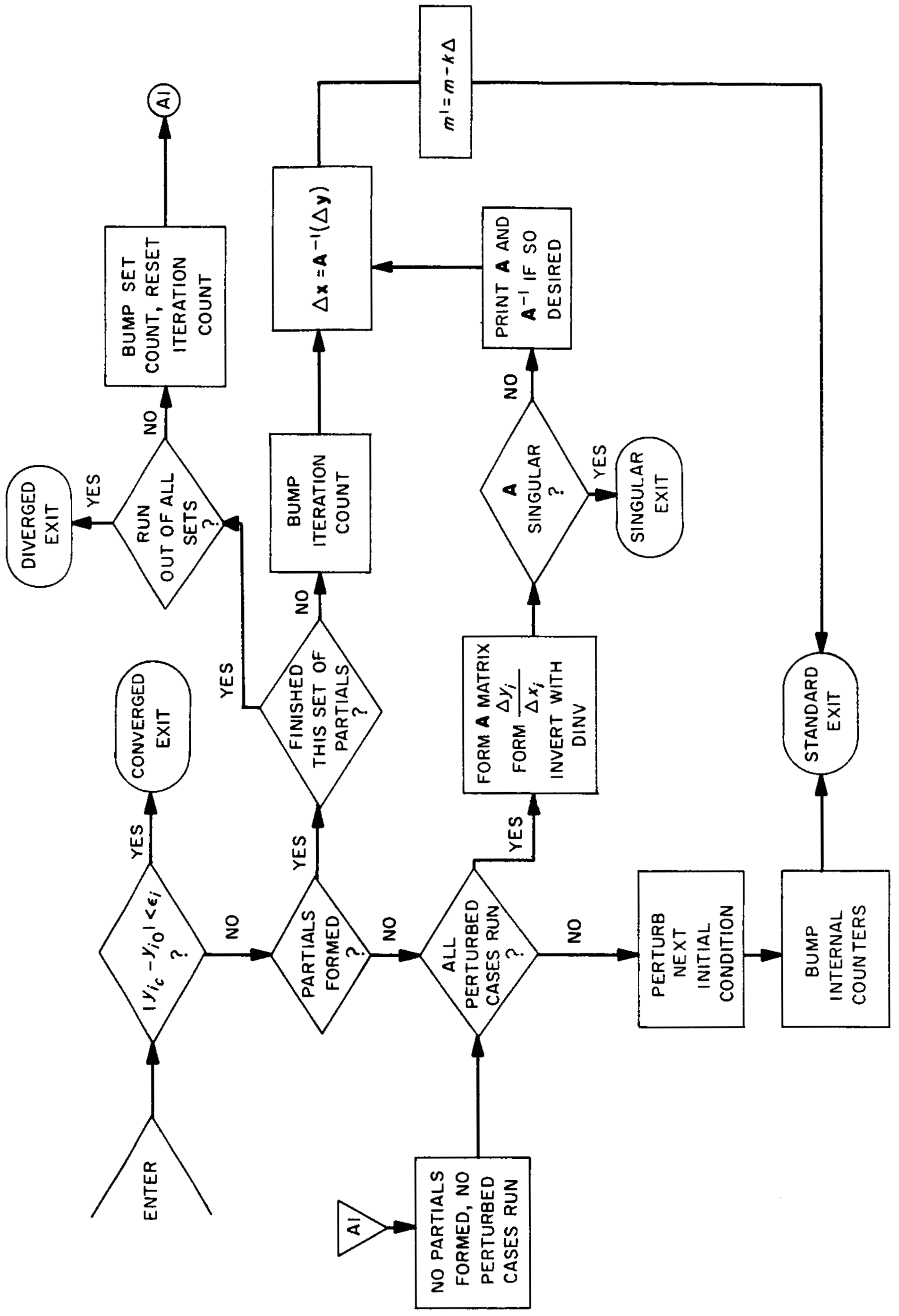
Because of midcourse-maneuver requirements, a few capabilities have been added. On option, a user may elect to use not *a priori* end conditions but an altered form of the ones computed on the first nominal trajectory. If a non-zero number θ_i is input, SEARCH sets $y_{t_i} = y_{0_i} + \theta_i$. In words, the i^{th} target value is automatically taken to be the first computed value of that particular variable plus the increment θ_i. This is done for any or all of the y_{t_i}'s.

For ease in maneuver computation, an auxiliary calculation, though not directly connected with the search procedure, is included. For each case about to be run, it involves the magnitude $|\mathbf{x}_j - \mathbf{x}_0|$ of the difference between the current set of initial conditions and those for the first nominal case. If $\Delta = |\mathbf{x}_j - \mathbf{x}_0|$ and if k and m are input quantities, the calculation performed is

$$m' = m - k\Delta.$$

(This is used in computing a mass change for a probe due to fuel consumption during burn.)

Figure 2 is a flow chart of the computation in SEARCH.



ENTER
$|y_{i_c} - y_{i_0}| < \epsilon_i$?
YES
CONVERGED EXIT
NO
PARTIALS FORMED ?
YES
FINISHED THIS SET OF PARTIALS ?
YES
RUN OUT OF ALL SETS ?
YES
DIVERGED EXIT
NO
BUMP SET COUNT, RESET ITERATION COUNT
A1
NO
BUMP ITERATION COUNT
$\Delta x = A^{-1}(\Delta y)$
$m' = m - k\Delta$
NO
ALL PERTURBED CASES RUN ?
YES
FORM A MATRIX FORM $\frac{\Delta y_i}{\Delta x_i}$ INVERT WITH DINV
A SINGULAR ?
YES
SINGULAR EXIT
NO
PRINT A AND A^{-1} IF SO DESIRED
NO
PERTURB NEXT INITIAL CONDITION
BUMP INTERNAL COUNTERS
STANDARD EXIT
A1
NO PARTIALS FORMED, NO PERTURBED CASES RUN

Fig. 2. Flow chart of computation in SEARCH

III. INPUT TO SEARCH

SEARCH operates under JPTRAJ and is controlled from a standard source deck.

1. There are three non-standard exits on the program card:

   a. The first non-standard exit is taken immediately upon the discovery of convergence. This is the *converged* exit.

   b. The second non-standard exit is called the *diverged* exit. It is taken when all iterations have run and there has been no convergence.

   c. The third non-standard exit will be taken when the matrix of partials **A** has zero determinant. This is called the *singular* exit.

   SEARCH always exits to the next sequential program in the source deck (i.e., takes normal return in JPTRAJ) except when one of the above three exits is warranted.

2. SEARCH is a Class 2 program under JPTRAJ. It may (and almost always will) lie within the range of a "backwards" GO.

3. The input data symbols and explanations appear in Table 1.

Table 1. Input symbols for SEARCH

Note: All cells below linked by "..." occupy sequentially ascending core locations.

	Purpose	Name	Mode	Remarks
1.	Independent variables of the search	IV1, IV2, ..., IV6	Floating	Given to next program via WANT card.
2.	Dependent variables of the search	DV1, DV2, ..., DV6	Floating	Always obtained via WANT cards.
3.	Perturbative increments used to generate partials (**A** matrix)	DELI1, ..., DELI6	Floating	The δi's
4.	Permissable error for each variable at target	EPSD1, ..., EPSD6	Floating	The vector $\boldsymbol{\varepsilon}$
5.	Actual target values	CVRG1, ..., CVRG6	Floating	$\mathbf{y}_T$
6.	Names of independent variables	IV1N, ..., IV6N	BCD	
7.	Names of dependent variables	DV1N, ..., DV6N	BCD	
8.	Number of iterations per set of partials	ITER	Fixed	Standard value is 3.
9.	Number of sets of partials before diverged exit is taken	NPR	Fixed	Standard value is 3.
10.	Data to be preserved between runs	STATS	– –	Data must always be saved. See below on how to do this.
11.	Flag to print **A** matrix	APRNT	Fixed	0 = suppress 1 = print
12.	Flag to print $\mathbf{A}^{-1}$ matrix	AVPRT	Fixed	0 = suppress 1 = print
13.	Flag to use analytic partials	AP	Fixed	0 = compute numerical partials 1 = use analytic partials
14.	Flag to print new initial conditions	IFLAG	Fixed	0 = suppress 1 = print
15.	Partials (**A** matrix)	PRTLS	Floating	Stored forward, rowwise
16.	Number of independent (and dependent) variables	ORDER	Fixed	The n in the expression "n by n search"
17.	The constant in the auxiliary calculation	CONST	Floating	The k in $m' = m - k\Delta$
18.	The initial m in auxiliary calculation	MASS1	Floating	The m in $m' = m - k\Delta$
19.	Increments to use with computed target value	CH1, CH2, ..., CH6	Floating	The θ_i's; ith target value is computed only if $\theta_i \neq 0$. Canned value is 0.
20.	Flag to compute partials only	PAR	Fixed	Standard value is 0. If PAR $\neq$ 0, no search is made. Instead, first matrix of partials is computed and then converged exit is taken.

The rules for deck setup are listed below:

1. SEARCH is an iterative program and will normally be read in from the JPTRAJ program area everytime it is to be used (i.e., at the beginning of each iteration). Because of this, all intermediate data, such as iteration counters, intermediate end conditions, etc., will be lost unless saved by a WANT card.

   RULE: The *last* card for SEARCH itself must be

   ```
   *        WANT        (S),STATS,200
   ```

   where (S) is the symbol appearing in the location field of the program card for SEARCH.

2. Current initial conditions are *always* found in IV1, ..., and must be passed via a WANT card *in* the program that needs them.

3. The first card after the last program intended for the range of SEARCH should usually be

   ```
   *        GO        (X)
   ```

   where (X) is the location field symbol in the SEARCH program card. (The exception to this is where a program returns to SEARCH via a non-standard return.)

4. The dependent variables must always be obtained by a WANT card for SEARCH from the program generating them.

A portion of the actual source deck used in JPL's midcourse maneuver program is illustrated in Fig. 3. This portion involves SEARCH and the trajectory integrator SPACE. The intent is to converge on the desired $\mathbf{B \cdot R}$, $\mathbf{B \cdot T}$, and a computed flight time. A card-by-card breakdown of the source deck follows in Table 2.

```
                RADOPT=1.025E8,0,0,11.12,.09607                           C
*       WANT    SP,(TAPEX),6                                              C
                PAGBCD=(SPACE TO M/C INJECTION PERTURBED)                 C
*       WANT    A,(PERPHS,MARPH1+11),4,(PERPHS,MARPH2+11),4               C
* SCH   SEARCH  B,X,X                                                     C
                DELI1=.001,.001,.001 ORDER=3                              C
                IV1N=(DX),(DY),(DZ)                                       C
                DV1N=(B.R),(B.T),(TL)                                     C
                AVPRT=1 APRNT=1                                           C
                CVRG1=15590.,9000. ITER=3  NPR=6                          C
*       WANT    A,(TTOL,EPSD1+2),(DELTL,CH3)                              C
*       ETC     MASS1,CONST                                               C
                EPSD1=75.,75.                                             C
*       WANT    SP,(XOP+21,IV1),3                                         C
*       WANT    SP1,(BRC,DV1),(BTC,DV2),(TFLIND,DV3)                      C
*SCHX   WANT    SCH,STATS,200                                             C
*SP1    SPACE   X                                                         C
* SP1X                                                                    C
*       WANT    A,(T1,INJT),2                                             C
*       WANT    SP,(XOP,INJX),3                                           C
*       USE     SPX,SPY                                                   C
                MARPH2=-0 MARPH2+27=112011000000/8 MARPH2+6=(EARTH)       C.001
                TARBCD=(MARS)                                             C.001
                INJBCD=(EARTH)                                            C.001
                PAGBCD=(SPACE TO M/C INJECTION)                           C.001
                FAZFLG=-1                                                 C.001
                INJTYP=0                                                  C.001
                MARPH1+27=112011000000/8                                  C.001
                MARPH1+38=10000000000/8                                   C.001
                RADOPT=1.025E8,0,0,11.12,.09607                           C.001
$       WANT    SP,(TAPEX),6                                              C.001
                PAGBCD=(            SEARCH B.T,B.R,TL FOR M/C)            C
                MARPH1=-1           MARPH1+11=20000,0,20000,0             C
                MARPH2=-1 MARPH2+11=30000,0,30000,0                       C
                MARPH2+4=.200E9                                           C
                MARPH3+38=100/8                                           C
* SP1Y                                                                    C
*       WANT    SCH,(IV1,INJDX),3,(STATE),(MASS,RADOPT+5)                 C
*       GO      SCH                                                       C
*  B    DECPR   B,CA1                                                     C
*       WANT    A,B.R,B.T,T1,2,TIME,2,VCAP,ACA,PMAX                       C
```

Fig. 3. A typical source deck using SEARCH

Table 2. Explanation of source deck used in Fig. 3

```
   RADOPT  = 1.025E8, 0, 0, 11.12, .09607
*  WANT   SP, (TAPEX), 6
          PAGBCD = (SPACE TO M/C INJECTION PERTURBED)
*  WANT   A,(PERPHS, MARPH1 +11), 4, (PERPHS, MARPH2 +11), 4
```

These cards are shown for completeness and do not affect the operation of SEARCH. They are some data and WANT cards for the immediately preceding program.

2.	*	SCH	SEARCH	B ,	X ,	X
	Indication of control card	location symbol used to refer to this program	The name of this program	First non-standard return. When (or if) convergence is reached, next program to execute is at location B (at bottom of Fig. 3).	Second non-standard return. If search fails to converge, control goes to location X, not shown in Fig. 3.	Third non-standard return. Control goes to location X if singular exit occurs.

3. DELI1 = .001, .001, .001	ORDER = 3
The symbolic name by δ_i, the amount by which each independent variable is to be separately perturbed to compute partials	This is a 3 by 3 search

Table 2 (Cont'd)

4. IVIN = (DX), (DY), (DZ)

The names for the three independent variables $\dot{x}$, $\dot{y}$, $\dot{z}$ are to be listed as DX, DY, and DZ.

5. DVIN = (B.R), (B.T), (TL)

These are the BCD names of the dependent variables.

6. AVPRT = 1	APRNT = 1
The user wants the A^{-1} matrix printed when computed	The user wants to print the A matrix

7. CVRGI = 15590., 9000.	ITER = 3	NPR = 6
These are the target values of B·R and B·T, respectively.	Once the partials are formed we make 3 iterations using them before making another set	Six sets of partials are made before we take the divergent exit.

8. * WANT	A ,	(TTOL, EPSDI + 2),	(DELTL, CH3)
Some of SEARCH's input comes from other programs.	The program supplying data is at location A, not shown in Fig. 3.	Tolerance on flight time(ϵ_3)	Increment used in computing target value for flight time

9. * ETC	MASSI, CONST
Continuation of the previous WANT card.	These 2 items are the inputs to the auxiliary equation.

Table 2 (Cont'd)

10. `EPSD1 = 75., 75.`

These numbers represent the tolerances for the miss for **B·R** and **B·T**

11. `* WANT SP, (XOP+21, IV1), 3`

This card inputs the original values of the independent variables, which will be used to run the nominal trajectory. On subsequent entries, SEARCH manages to overlook this card and maintains the proper injection conditions in IV1, ... The location SP lies elsewhere in the source deck and is not shown in Fig. 3.

12. `* WANT SP1, (BRC, DV1), (BTC, DV2), (TFLIND, DV3)`

When the subsequent SPACE (starting at card 14) has run and return is made to SEARCH, this card gets the three variables just computed (**B·R**, **B·T**, t_l).

13. `* SCHX WANT SCH, STATS, 200`

Location SCH is the location of the program card for SEARCH. This card saves 200 words of STATS at the end of every run and recalls them at the start of the next iteration. This card is mandatory at the end of every SEARCH data deck.

Table 2 (Cont'd)

14. * SPI SPACE X

This is the program card for the trajectory integrator and is the program over which SEARCH will iterate.

15.
```
*  SPIX
*       WANT     A,(T1,INJT),2
*       WANT     SP,(XOP,INJX),3
*       USE      SPX,SPY
                 MARPH2=-0 MARPH2+27=112011000000/8 MARPH2+6=(EARTH)
                 TARBCD=(MARS)
                 INJBCD=(EARTH)
                 PAGBCD=(SPACE TO M/C INJECTION)
                 FAZFLG=-1
                 INJTYP=0
                 MARPH1+27=112011000000/8
                 MARPH1+38=100000000000/8
                 RADOPT=1.025E8,0,0,11.12,.09607
$       WANT     SP,(TAPEX),6
                 PAGBCD=(             SEARCH B.T,B.R,TL FOR M/C)
                 MARPH1=-1           MARPH1+11=20000,0,20000,0
                 MARPH2=-1 MARPH2+11=30000,0,30000,0
                 MARPH2+4=.200E9
                 MARPH3+38=100/8
*  SPIY
```

These cards are miscellaneous inputs for SPACE.

16. * WANT SCH, (IVI,INJDX), 3, (STATE), (MASS,RADOPT+5)

* WANT	SCH,	(IVI,INJDX), 3,	(STATE),	(MASS,RADOPT+5)
This card obtains from SEARCH the necessary computed quantities.	Location symbol of SEARCH	3 independent variables, representing $\dot{x}$, $\dot{y}$, $\dot{z}$.	The flag cell STATE. (Its use is described in the section on output.)	Output from auxiliary equation $m' = m - k\Delta$

Table 2 (Cont'd)

17.
```
*       GO    SCH
```
This card returns execution to SEARCH after SPACE has finished. Note that this card is bypassed by the B in card No. 2 above.

18.
```
*  B   DECPR   B,CAI
*      WANT    A,B.R,B.T,TI,2,TIME,2,VCAP,ACA,PMAX
```
These two cards are a continuation of the source deck. The program DECPR is executed only after SEARCH has converged.

IV. OUTPUT

SEARCH has two blocks of output, each printed at a different time. The first is printed at the initial entry; it is merely a rehashing of the input data. The other block appears at subsequent entries. The output from the input example is illustrated on the following pages.

The subroutine used for output is PROUT, the general SFOF output routine. Output is always put on tape A3 for off-line printing. In Mode 2 it also appears on the SC-3070 printer.

The output symbols used in SEARCH represent computed values intended for other programs. They are specified in Table 3.

The output from the sample case in the section on input is illustrated in Figs. 4 and 5. Figure 4 shows the output from the initial entry to SEARCH, and Fig. 5 shows the output from the next four entries. A line-by-line breakdown of the output follows in Table 4.

```
                                  SEARCH PROGRAM PARAMETERS
    INDEPENDENT VARIABLE NAMES...      DX          DY          DZ
      DEPENDENT VARIABLE NAMES...      B.R         B.T         TL
      TARGET VALUES         +.15590000E+05  +.90000000E+04  +.00000000E+00
T. V. MODIFICATIONS         +.00000000E+00  +.00000000E+00  +.70333007E+00
PERMISSIBLE ERROR VECTOR    +.75000000E+02  +.75000000E+02  +.50000000E-02
GENERATIVE INCREMENTS       +.10000000E-02  +.10000000E-02  +.10000000E-02
     THE ORDER IS  3, WITH  6  SETS OF PARTIALS OF  3  ITERATIONS EACH.
```

Fig. 4. Output from the initial entry to SEARCH

Table 3. Output symbols for SEARCH

Note: Symbols linked by "..." indicate they occupy sequentially ascending core locations.

Purpose	Name	Mode	Remarks
1. Current values of independent variables	IV1, IV2, ..., IV6	Floating	Initially contain input values, but subsequently contain "updated" values.
2. Information about type of case about to be run.	STATE	--	Sign: + = nominal or applied case; – = perturbed case Address: ordinal number of applied or perturbed case about to run (1st, 2nd, etc.) Decrement: ordinal number of set of partials in use or being computed STATE is good for print control, etc., in subsequent programs.
3. Number of the return just taken	FLAG	Fixed	Contains 0 for normal return, 1 for 1st non-standard return, etc. Number is right-adjusted in address. This number is useful if a program has to know which return was taken.
4. Output value of auxiliary equation.	MASS	Floating	The m' in $m' = m - k\Delta$.

Table 4. Description of output shown in Fig. 4

1.
```
SEARCH  PROGRAM  PARAMETERS
```
This is the heading that always appears on the initial printout.

2.
```
INDEPENDENT  VARIABLE  NAMES...  DX   DY   DZ
```
These names are a printout of the quantities IV1N, IV2N, and IV3N. This is the only place in which they are used.

3.
```
DEPENDENT  VARIABLE  NAMES...  B.R   B.T   TL
```
These are the quantities DV1N, DV2N, and DV3N. Here, also, is the only place they are used.

4.
```
TARGET  VALUES   +.15590000E +05
  +.90000000E+04     +.00000000E+00
```
By examining the source deck in Figure 1, one can see that the first two numbers above are CVRG1 and CVRG2. Since the "compute option," CH3, is used for flight time target value, the last number above is not yet known and is printed as zero.

5.
```
T.V. MODIFICATIONS +.00000000E+00
  +.00000000E+00    +.70333007E+00
```
This line is the printout of the increments used in computing target values. This line is printed only if at least one of them is non-zero. The target value is computed only for CVRG3 in this example.

6.
```
PERMISSIBLE  ERROR  VECTOR  +.75000000E+02
  +.75000000E+02    +.50000000E-02
```
This line represents the printout of the vector ε input as EPSD1, EPSD2, and EPSD3.

7.
```
GENERATIVE  INCREMENTS  +.10000000E-02
  +.10000000E-02    +.10000000E-02
```
The DEL11, etc., are listed in this line of print.

8.
```
THE ORDER IS    3,  WITH   6  SETS OF PARTIALS OF  3  ITERATIONS EACH
```
The items in this line are: ORDER NPR ITER

Continuing with our output example, the reader should examine Fig. 5. It is an illustration of SEARCH's output on *four* subsequent reentries. The print is divided into four blocks. The first block was printed after the nominal case had run, and each of the remaining three after a perturbed case. The first line of every block is

```
SEARCH HAS BEEN ENTERED AGAIN **********
```

This is printed only as an indication of progress through the program. The second line of every block consists of three unlabeled numbers. These are the just-computed values of DV1, DV2, and DV3 (the dependent variables). Originally intended only for checkout, they remain there to this day.

Block 4 contains additional information. It is the print after the final perturbed case has run. At this point the partials are computed and printed. After printing the $\mathbf{A}^{-1}$ matrix SEARCH is prepared to exit and run its first applied case. Prior to doing so two sets of information are written: (1) the $\Delta\, \mathbf{x}_i$ vector, which is the computed change to the previous set of initial values, and (2) the total increments, which are the difference between the present initial values and the original nominal ones.

Incidentally, each time before SEARCH was reentered, the program SPACE had to run. Normally SPACE will produce massive output. By profitably checking the polarity of the cell STATE, SPACE was able to eliminate entirely a lot of unnecessary print.

```
BLOCK
      SEARCH HAS BEEN ENTERED AGAIN **********
  1
      0.32387364E 05  0.14927771E 06  0.23759727E 03

      SEARCH HAS BEEN ENTERED AGAIN **********
  2
      0.28153972E 05  0.13518422E 06  0.23771282E 03

      SEARCH HAS BEEN ENTERED AGAIN **********
  3
      0.36529507E 05  0.17690161E 06  0.23749227E 03

      SEARCH HAS BEEN ENTERED AGAIN **********

      0.33070222E 05  0.16180478E 06  0.23754819E 03

              THE MATRIX OF PARTIALS IS

         -0.42333927E 07  0.41421437E 07  0.68285716E 06

         -0.14093482E 08  0.27623902E 08  0.12527074E 08

          0.11554527E 03 -0.10499953E 03 -0.49085616E 02
  4
              THE INVERSE PARTIAL MATRIX IS

         -0.19035117E-07  0.61710586E-07  0.15484267E-01

          0.35429292E-06  0.60434265E-07  0.20352112E-01

         -0.80267938E-06  0.15988319E-07 -0.27458726E-01

       I.V. INCREMENTS (DELTAS) ARE  0.25536559E-02 -0.11453521E-03 -0.80724273E-02

      TOTAL INCREMENTS ARE  0.25536716E-02 -0.11454523E-03 -0.80724359E-02
```

Fig. 5. Output from subsequent entries to SEARCH

V. SUBROUTINES

The "main" program is entered from JPTRAJ and prints out the initial entry data. On a normal entry the dependent variables received from WANT cards are transferred by the main program and control is then transferred to the subroutine SRCH.

The other principal routine used is DINV, a double-precision matrix-inversion subroutine. The write-ups to SRCH and DINV are included in this section.

Below is a list of secondary subroutines used in the program. They are standard routines for SFOF operation and their write-ups may be obtained from the JPL systems office or the SFOF programming group.

PROUT

OUTUS

IOCS (*SFOF version*)

(DFAD) (7094 *hardware version*)

A. SRCH

SRCH is the principal subroutine of the JPTRAJ program SEARCH. It performs the analysis of end conditions and the matrix inversions essential to the Newton-Raphson iteration method. This subroutine shares a buffer of 200 words of COMMON storage which is properly maintained by the main program of SEARCH. The buffer contains information that must be preserved by a WANT card.

The equations in the SRCH subroutine are just those described in the method given in the SEARCH program write-up. The only operation not explicitly done by SEARCH is matrix inversion. All parameters are communicated through COMMON, so the calling sequence is simply

CALL SRCH

SRCH uses 600_{10} words of lower core and 200_{10} words of common storage. It is written in the FORTRAN language.

B. DINV

DINV solves the matrix equation $\mathbf{AX} = \mathbf{B}$, where $\mathbf{A}$ is a square coefficient matrix and $\mathbf{B}$ is a matrix of constant vectors. $\mathbf{A}^{-1}$ is also obtained; indeed, inversion may be the sole aim in a particular usage. Finally, the determinant of $\mathbf{A}$ is available. Jordan's method is used to reduce a matrix $\mathbf{A}$ to the identity matrix $\mathbf{I}$ through a succession of elementary transformations: $t_n \, t_{n-1} \cdots t_1 \, \mathbf{A} = \mathbf{I}$. If these transformations are simultaneously applied to $\mathbf{I}$ and to a matrix $\mathbf{B}$ of constant vectors, the result is $\mathbf{A}^{-1}$ and $\mathbf{X}$ where $\mathbf{AX} = \mathbf{B}$. Entrance is made via the FORTRAN statement in the calling program:

```
CALL DINV (A,N,B,M,DETERM)
```

where

1. N is the order of A; $N \leq 6$.
2. M is the number of column vectors in B.
3. DETERM is the location in which the determinant is to be placed.

Suitable variable names may replace the dummy variables listed above. For compatibility purposes with the subroutine which was compiled on the basis of $N = 6$, dimension statement entries for the two-dimensional arrays, A and B, in the calling program must have row dimension equal to 6, e.g., A(6,6), or B(6,2). (If this imposes too severe a storage requirement, or, on the other hand, if it is desired that $N > 6$, the subroutine can be recompiled with a new DIMENSION statement replacing all entries of 6 by the desired value.) At the return to the calling program, $\mathbf{A}^{-1}$ is stored at A and $\mathbf{X}$ at B. $M = 0$ or negative signals that the routine is to be used solely for inversion; note, however, that in the CALL statement an entry corresponding to B must still be present.

Space required: $711_8(457_{10})$ locations in addition to an extent of $60 + N$ locations at COMMON distributed as follows down from upper memory:

1. N locations for PIVOT – the array of pivot elements used in the inversion.
2. $6 - N$ locations not used.
3. N locations for column 1 of INDEX – a 2-column array which records consecutive row interchanges.

4. 6 – N locations not used.

5. N locations for IPIVOT – an array used to prevent duplicate pivotings on any single row.

VI. ITER

A. Method

Suppose a user wants to run m cases of a program and vary n selected parameters between cases. There are two methods of using ITER to accomplish this. The first method is to input all $m \cdot n$ initial conditions into ITER. ITER will then present them, n at a time, in a buffer which may be reached by a WANT card for the program needing them. This is type 1 iteration.

The other method is that one set of n initial conditions along with n increments may be input into ITER. At the completion of every run ITER will add each increment to the corresponding previous initial condition and present the results as new initial conditions. This is type 2 iteration.

Mathematically speaking, for type 1 iteration a user inputs an m by n matrix of initial conditions

$$[a_{ij}]$$

in rowwise fashion and ITER outputs the a_{ij}'s one row at a time. For type 2 iteration only the first row is input along with n increments δ_j, $j = 1, 2, \cdots, n$. Upon the k^{th} entry to ITER the next row of the matrix is found by

$$a_{kj} = a_{k-1,j} + \delta_j, \; j = 1, 2, \cdots, n.$$

In some cases a user will select several values for each of two parameters, x and y, and will want to run a program using all values of y for each and every value x. This may be easily done by nesting ITER – that is, by iterating over ITER itself, like nested DO loops in FORTRAN. An example in the section on input illustrates nesting.

B. Input

Table 5 is a description of the input to ITER.

1. All initial conditions for either type of iteration get input into the buffer beginning with VARY.

2. Current initial conditions for the program iterated over will always be in OUTPUT.

3. 52 words of the buffer TABL in ITER must always be saved by a WANT card. This buffer contains current initial conditions and counters. The cell TABL itself contains the current case number in the decrement and the product of *n* and the case number in the address. Its contents change between cases, so it is useful for the cell RESET in nested iterations.

4. ITER is a class 2 program under JPTRAJ, and return to it is usually made by a GO card.

5. ITER has one non-standard return and uses it when complete.

Below are two contrived source decks showing how ITER can be used. The other program used in the examples is NECON, JPL's Near-Earth Conic Program. No data cards for NECON are shown, to keep the illustrations short. To each card is appended descriptive text.

CASE 1

The first example shows a source deck intended to run five cases of NECON, varying the launch azimuth between cases. The azimuths to be used are 96°, 99°, 102°, 105°, and 108°.

(1)	`* A  ITER  X`	This is the program card for ITER. When finished, ITER exits at location X below.
(2)	`N = 1  SETS = 5`	There are five cases to run, with one item changing per case.
(3)	`VARY = 96.,99.,102.,105.,108.`	These are the five azimuths which are to be used.
(4)	`TYPE = 1`	Type 1 iteration – all parameters are input on cards.
(5)	`* WANT  A,TABL,52`	Mandatory at the end of every ITER data package.
(6)	`* NECON  X`	Program card for NECON.
(7)	`* WANT  A,(OUTPUT,AZ)`	ITER has the proper value for azimuth in the cell OUTPUT.
(8)	`* GO  A`	When NECON finishes, control returns to ITER, at location A.
(9)	`* X  END`	End of source deck.

If the two cards

```
VARY = 96.  DELTA = 3.
TYPE = 2
```

replace cards (3) and (4) above, the result is a source deck specifying type 2 iteration. The initial value of 96 deg and the increment DELTA added four successive times generate the five required azimuths.

CASE 2

The remaining source deck is an illustration of nested iteration. Five launch azimuths are run on each of 30 successive days, giving a total of 150 runs of NECON.

As stated in the setup instructions the cell TABL changes from case to case. This is a useful control for the cell RESET and the consequent restarting of the nested ITER.

(1)	`* A ITER X`	Program card for "outside" ITER.
(2)	`TYPE = 2  SETS = 30  N = 1`	Type 2 iteration, 30 cases (1 case = 1 day), changing 1 element per case.
(3)	`VARY = 5730.  DELTA = 1.`	Initial date value is 5730, and it will change by 1 from case to case.
(4)	`* WANT A,TABL,52`	Save 52 words for ITER itself between cases.
(5)	`* B ITER Y`	Program card for nested ITER.
(6)	`TYPE = 2  SETS = 5  N = 1`	This iteration is over azimuths. It is type 2, five cases, varying one item per case
(7)	`VARY = 96  DELTA = 3.`	Azimuth is initially 96 deg and is changed by 3 deg for each case.
(8)	`* WANT A,(TABL,RESET),1`	When "outside" ITER changes, nested ITER restarts. Outside case will not change until all five nested cases are run.
(9)	`* WANT B,TABL,52`	Data for this case of ITER must be preserved between runs.
(10)	`* NECON X`	Program card for NECON.
(11)	`* WANT A,(OUTPUT,DAY)`	Day number is located in outside ITER.
(12)	`* WANT B,(OUTPUT,AZ)`	Azimuth is located in nested ITER.

(13)	* GO B	When nested ITER completes one case, control returns to it for next case.
(14)	* Y GO A	When nested ITER finishes fifth azimuth, control comes here and hence to the outside ITER. The day number is increased by 1 and iteration resumes.
(15)	* X END	When the outside ITER finishes, it comes here to exit.

There is no limit to the depth of nesting. The user must be sure of which variables belong to each level, and the cell TABL in each level must be put into the cell RESET at the next lower level.

There is no printed output from ITER.

Table 5. Input symbols for ITER

Function	Symbol	Mode	Remarks
1. The rowwise elements of the $[a_{ij}]$ matrix	VARY	Fixed or Floating	$m \cdot n$ values for type 1; n values for type 2
2. Number of cases to run	SETS	Fixed	This number is the value of m.
3. Number of parameters per case	N	Fixed	N $\leq$ 50 (Note: SETS*N $\leq$ 2500.)
4. Type of iteration	TYPE	Fixed	1 = type 1, 2 = type 2 (Canned value is 1)
5. Increments used in type 2 iteration	DELTA	Same as for VARY	These are the δ_j's, where $j = 1, 2, \cdots, n$.
6. Buffer containing next initial conditions	OUTPUT	---	Can hold at most 50 at one time.
7. Flag to restart	RESET	Anything	If the contents of RESET change between cases, ITER restarts. (Used for nesting.)
8. Status record	TABL	---	Must be preserved by WANT card from case to case. See setup instructions.

ACKNOWLEDGMENT

The subroutine DINV is a modification of a routine done originally by Mr. B.S. Garbow of the Argonne National Laboratory, Lemont, Illinois.