HEWLETT
PACKARD

HP 82484A

Curve Fitting Pac

Owner's Manual

For the HP-71

Notice

Hewlett-Packard Company makes no express or implied warranty with regard to the keystroke procedures and program material offered or their merchantability or their fitness for any particular purpose. The keystroke procedures and program material are made available solely on an "as is" basis, and the entire risk as to their quality and performance is with the user. Should the keystroke procedures or program material prove defective, the user (and not Hewlett-Packard Company nor any other party) shall bear the entire cost of all necessary correction and all incidental or consequential damages. Hewlett-Packard Company shall not be liable for any incidental or consequential damages in connection with or arising out of the furnishing, use, or performance of the keystroke procedures or program material.

HEWLETT
PACKARD

Curve Fitting Pac

Owner's Manual

For Use With the HP-71

March 1984

82484-90001

Printed in Singapore

Introducing the Curve Fitting Pac

The HP 82484A Curve Fitting Pac is a powerful tool that enables you to perform functions that are not common to portable computing devices. The Curve Fitting Pac permits you to:

- Fit a general model function (linear or non-linear) to a set of data using the `CFIT` program.
- Determine local maxima and minima of a large class of real-valued functions using the `OPTIMIZE` program.

The Curve Fitting Pac allows you to quickly and easily choose a model, fit a curve to your data, choose another model, and fit the curve again—all in a matter of seconds.

Features of the pac include:

- A matrix editor that makes entering and editing data easy.
- A built-in library of commonly-used fit models.
- The ability to store data and then retrieve it for later use.
- The ability to direct all intermediate and final results to a peripheral printer.

Contents

How to Use This Manual

The information in this manual assumes that you have read sections 1, 6, 8, 9, and 11-14 in your *HP-71 Owner's Manual* and that you are familiar with the following HP-71 operations: keyboard operation, file operations, writing and running simple subprograms, manipulating data, using flags, and correcting error conditions. In addition, if you plan to use a printer with this pac, you should know how to install and use a printer using the HP 82401A HP-IL Interface.

This manual is both a learning and reference tool. Read through section 1, "Getting Started," for an overview of what the Curve Fitting Pac can do. Then read section 2, "Curve Fitting," or section 3, "Optimizing a Function," depending on which application you want to use first. Later, if you need descriptions or reminders on how parts of the programs work, you can use these sections for reference. At the end of section 2, you will find "A CFIT Example." This is a comprehensive example for you to key in to get familiar with the main program features.

There are also several appendixes for your reference:

- Appendix A, "Owner's Information," includes warranty and service information.
- Appendix B is "Error and Status Messages." If CFIT or OPTIMIZE cannot carry out an operation, an error message will be generated. Refer to this appendix for an explanation of the messages the Curve Fitting Pac can generate.
- Appendix C, "Numerical Methods," describes the mathematics used in this pac and discusses situations that are difficult for the program to handle.
- Appendix D is "User-Accessible Routines." Here you will find the subprograms in the pac that you can access and how they interface with the main programs and each other.
- Appendix E is "Library Subprograms." CFIT provides a library of model subprograms for some commonly used models. Refer to this appendix for a list of these subprograms.
- Appendix F, "Applications File Format (HPAF)," describes the special format that this HP-71 application uses to store fit data.
- Appendix G is "Creating Your Own Model or Function Subprogram." You'll find information on syntax, calling relationships, and memory requirements, plus two examples of typical user-written subprograms here.
- Appendix H is a list of the file names used by the programs in this pac.
- Appendix I is a short glossary of terms used in this manual.

At the end of the manual is a complete subject index.

Getting Started

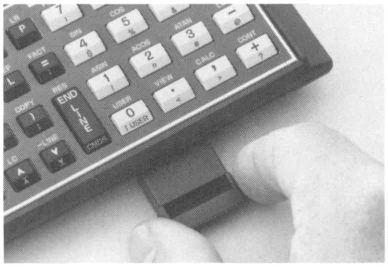
Installing and Removing the Curve Fitting Module

The curve fitting module can be plugged into any of the four ports on the front edge of the HP-71.

CAUTIONS

- Be sure to turn off the HP-71 (press [f] [ON]) before installing or removing any module.
- Whenever you remove a module to make a port available for another module, be sure to turn the HP-71 on and then off *while the port is empty* before installing the new module.
- Do not place fingers, tools, or other foreign objects into any of the ports. Such actions could result in minor electrical shock hazard and interference with pacemaker devices worn by some persons. Damage to port contacts and internal circuitry could also result.

To insert the curve fitting module, orient it so that the label is right-side up, hold the HP-71 with the keyboard facing up, and push in the module until it snaps into place. During this operation be sure to observe the precautions described above.



To remove the module, use your fingernails to grasp the lip on the bottom of the front edge of the module and pull the module straight out of the port. Install a blank module in the port to protect the contacts inside.

What the Curve Fitting Pac Does

The Curve Fitting Pac provides two main capabilities:

- The ability to fit a general model function (linear or non-linear) with up to 20 unknown parameters to a set of data.
- The ability to determine local minima (or maxima) of a large class of real-valued functions with up to 20 variables.

Both of these applications are based on the implementation of a powerful algorithm introduced by R. Fletcher and M.J.D. Powell. The method is known as the Fletcher-Powell Method (hereafter referred to as the FP Method) and is explained in detail in appendix C.

A printer or a video display is not required to use this pac since all data and output can be directed to the HP-71 display. You probably will find a printer or video display useful, however, for viewing large amounts of stored data as well as intermediate and final results. If you have an HP 82401A HP-IL Interface and a compatible printer or video display, the programs will print (or display) the output in an easily understood format. If you want a printer or video display and don't have one, contact your authorized Hewlett-Packard dealer for information on printers, video displays, and the HP 82401A HP-IL Interface.

Using CFIT

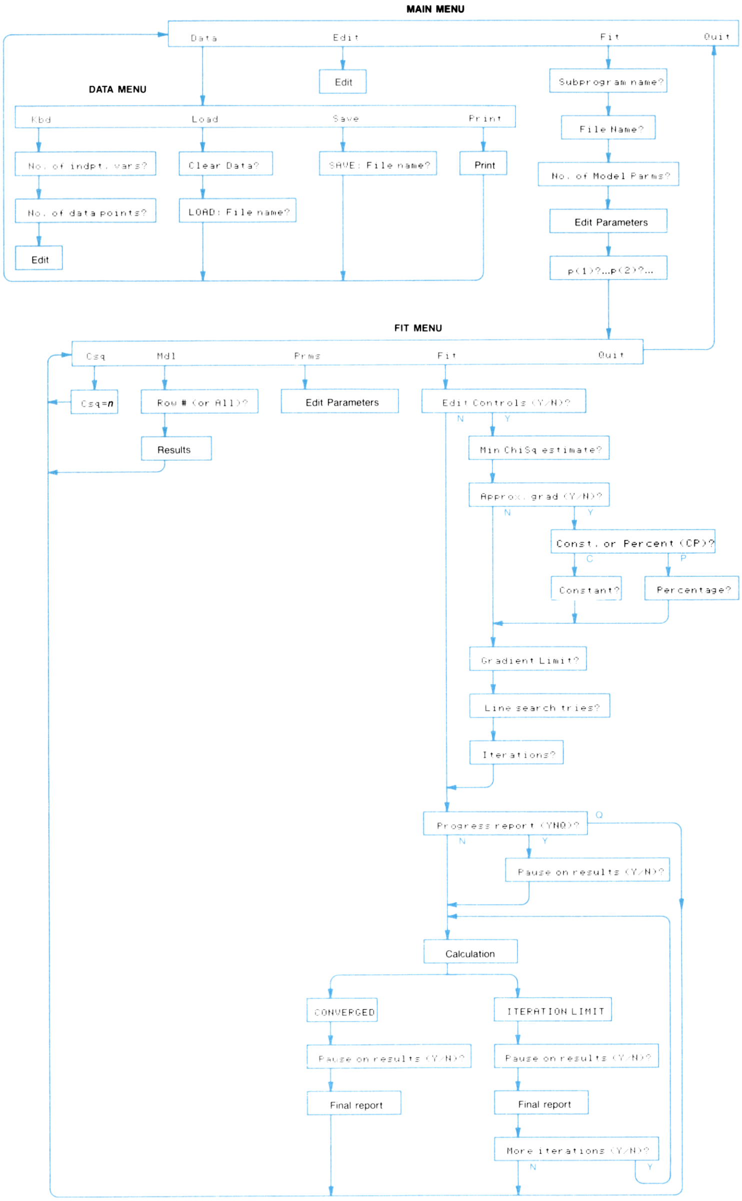
Overview

When you run the program CFIT, there are six main steps to go through to fit a curve to your data:

1. Create a model subprogram (or select one from the built-in library of model subprograms).
2. Enter the data points and optionally save them in a file.
3. Specify the name and location of your model subprogram.
4. Specify an initial guess for the model parameters.
5. Optionally do one or more of the following:
   - Compute the Chi Square value using the current model parameters.
   - Edit the model parameters.
   - Edit the program control values.
6. Fit the curve.

Flow Chart of CFIT Menus

What follows is a flow chart of the steps involved in and the menus associated with the curve fitting procedure.



CFIT Example: Stock Predicting

Without worrying too much about the meaning of all the keystrokes, key in this example and see how easily CFIT fits a curve to a set of data. What you will do in this example is:

1. Enter a set of data into a working array.
2. Enter an additional row of data to be evaluated but not used for the fit.
3. Save the data in a file.
4. Fit the curve and get the results.
5. Evaluate the model to predict the value of the unknown dependent variable corresponding to the data entered in step 2.

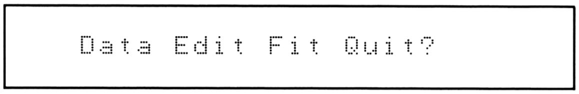
Example: The asking price for three stock issues was recorded at the close of trading on six successive Fridays and is given in the following table. Assuming the linear relationship $Y = aA + bB + c$ between the stock prices, what value can you expect for the price Y when $A = 42.5$ and $B = 28$?

Week	Stock *A*	Stock *B*	Stock *Y*	Weight
1	37.125	24.0	73.25	1
2	34.0	26.5	63.0	1
3	40.0	27.375	72.5	1
4	39.625	29.0	71.0	1
5	38.0	29.0	66.875	1
6	41.0	30.875	72.0	1
7*	42.5	28.0	?	Inf

* This row is the interpolate row. The weight of "Inf" causes this row to be ignored in the fit process.

Input/Result

RUN CFIT [END LINE]

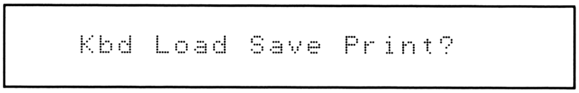


Data Edit Fit Quit?

The main menu.

[D]

You need to enter data from the keyboard, so press [D] to access the Data menu.



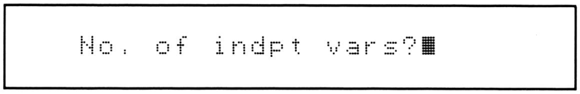
Kbd Load Save Print?

The Data menu.

[K]

Press [K] to select keyboard entry.

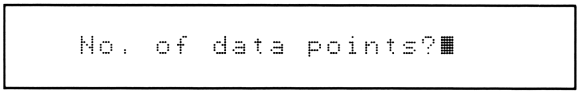
Input/Result



```
No. of indpt vars?■
```

2 END LINE

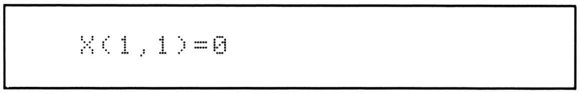
In this example there are two independent variables (*A* and *B*).



```
No. of data points?■
```

7 END LINE

There are seven data points, including the interpolate.

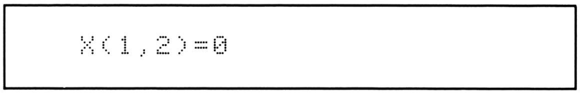


```
X(1,1)=0
```

The program branches to the editor and displays the first element of the first row of the array. All the values in the array default to 0 (except the weights, which default to 1).

37.125 END LINE

Enter the correct value of the first element from the table—stock *A*, week 1.

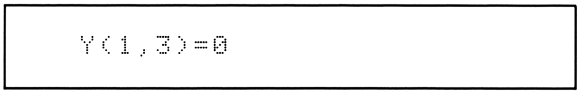


```
X(1,2)=0
```

The next element of the same row of the array is displayed. You can accept the default by pressing END LINE, or you can enter another value.

24 END LINE

Enter the value of the second element from the table—stock *B*, week 1.

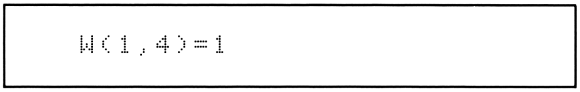


```
Y(1,3)=0
```

The program steps through the array one element at a time.

73.25 END LINE

Enter the value of the third element from the table—stock *Y*, week 1.

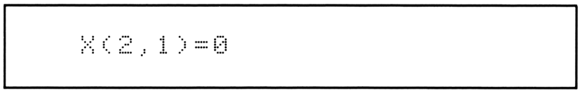


```
W(1,4)=1
```

The last element in this row of the array (the weight) is displayed for editing.

END LINE

Press END LINE to select the default weight of 1.



```
X(2,1)=0
```

The first element in the second row of the array is displayed.

Input/Result

34 END LINE

Enter the value for stock *A*, week 2.

```
X(2,2)=0
```

The program will continue to step through each element of the entire array.

26.5 END LINE

Enter the value for stock *B*, week 2.

```
Y(2,3)=0
```

63 END LINE

Enter the value for stock *Y*, week 2.

```
W(2,4)=1
```

The weight for the second row is displayed.

END LINE

Use the default weight of 1. Continue to enter the values from the table as the program calls for them.

⋮

```
X(7,1)=0
```

The program prompts for the first element in the last row (the interpolate row).

42.5 END LINE

The value of stock *A* to be used to solve for the unknown value of stock *Y*.

```
X(7,2)=0
```

28 END LINE

The value of stock *B* to be used to solve for the unknown value of stock *Y*.

```
Y(7,3)=0
```

This is the unknown value we are looking for.

END LINE

Let it default to zero.

```
W(7,4)=1
```

The weight for the interpolate is displayed.

g inf END LINE

While holding down the g key, type in inf. This gives this row a weight of infinity, thus effectively eliminating it from the fit process.

Input/Result

```
X(1,1)=37.125
```

The program has gone through the entire array and wraps back to the first element.

`Q`

Press this key to exit the editor and return to the main menu.

```
Data Edit Fit Quit?
```

`D`

Select the `Data` option to display the Data menu.

```
Kbd Load Save Print?
```

`S`

Press `S` to start the process for saving data to a file.

```
SAVE: File name?■
```

The program prompts for the file name to save the data into.

`STOCK` `END LINE`

Save it into a file called `STOCK`.

```
Saving...
Data Edit Fit Quit?
```

The program displays a message while the file is being created and the data copied to it. Then the main menu reappears.

`F`

Press `F` to start the fitting process.

```
Subprogram name?■
```

The program prompts for the name of the model subprogram.

`LIN` `END LINE`

`LIN` is the built-in linear model subprogram.

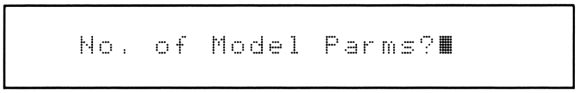
```
File name?■
```

Now `CFIT` asks for the name of the file that the subprogram is in.

`FITLIB` `END LINE`

`LIN` is in the library file `FITLIB`.

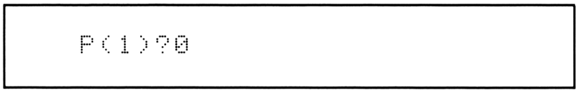
Input/Result



```
No. of Model Parms?■
```

3 END LINE

The model has three parameters (*a*, *b*, and *c*).

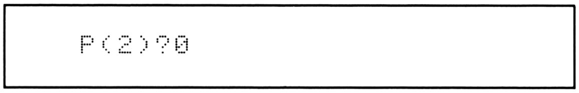


```
P(1)?0
```

Asks for a guess for the value of the first parameter.

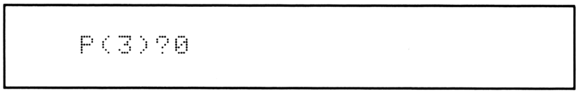
END LINE

Since you probably have no idea, use the default of 0. Do the same for the other parameters.



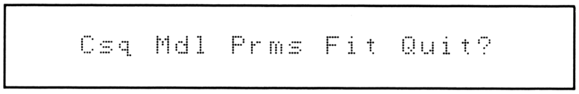
```
P(2)?0
```

END LINE



```
P(3)?0
```

END LINE

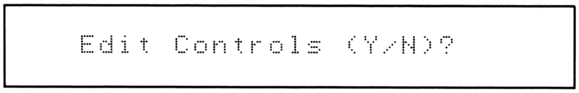


```
Csq Mdl Prms Fit Quit?
```

After the last guess is entered, the program displays the Fit menu.

F

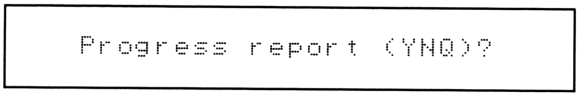
Press F to fit the curve.



```
Edit Controls (Y/N)?
```

N

For this example you do not need to edit the program controls; you can use the defaults.



```
Progress report (YNQ)?
```

The program asks if you want to view an iteration-by-iteration progress report.

N

Press N for no.

Input/Result

```
Working...
1 |Grd|: 5.86691E+001
2 |Grd|: 1.28785E+000
CONVERGED
```

Assuming you don't have a printer or video display attached, the norm of the gradient at each iteration is displayed. (If you do have a printer or video display attached, the results are output as shown in the example in section 2, on pages 47–49.)

END LINE

After the program converges, press END LINE to continue and see the results.

```
Pause on results(Y/N)?
```

Y

Press Y to pause between each result. This allows you to scroll through the results at a comfortable pace. Just press END LINE when you want to go on to the next one.

```
ChiSq:  3.4949031E+000
Percent: 32.100
|Grd|: 4.7886019E-006
P(1):  1.985300185E+000
P(2):-1.332683140E+000
P(3):  3.078786602E+001
G(1):-3.874125000E-006
G(2):-2.812775000E-006
G(3):-1.008000000E-007
Csq Mdl Prms Fit Quit?
```

As you scroll through the results, notice that P(1), P(2), and P(3) give you the values for *a*, *b*, and *c* in your linear model. G(1), G(2), and G(3) are the partials of Chi Square with respect to the model parameters. The other information will be described later.

M

Since your ultimate goal is to get the predicted value for stock *Y*, press M to begin evaluating the model at your interpolate.

```
Row # (or All)?■
```

The program asks if you want to evaluate one row or all rows.

7 END LINE

Evaluate row 7, the row with your unknown stock price.

Input/Result

```
Row 7: F = 77.8479959488
```

The model predicts 77.85 for stock *Y*.

[END LINE]

```
Csq Mdl Prms Fit Quit?
```

[Q]

Exit the curve fitting routine.

```
Data Edit Fit Quit?
```

[Q]

Quit the program.

```
Done
```

So, assuming a linear relationship Y = $aA + bB + c$ between stock prices, and given a price for stock *A* of 42.5 and stock *B* of 28, you can predict the price of stock *Y* to be 77.85, or 77⅞.

Using OPTIMIZE

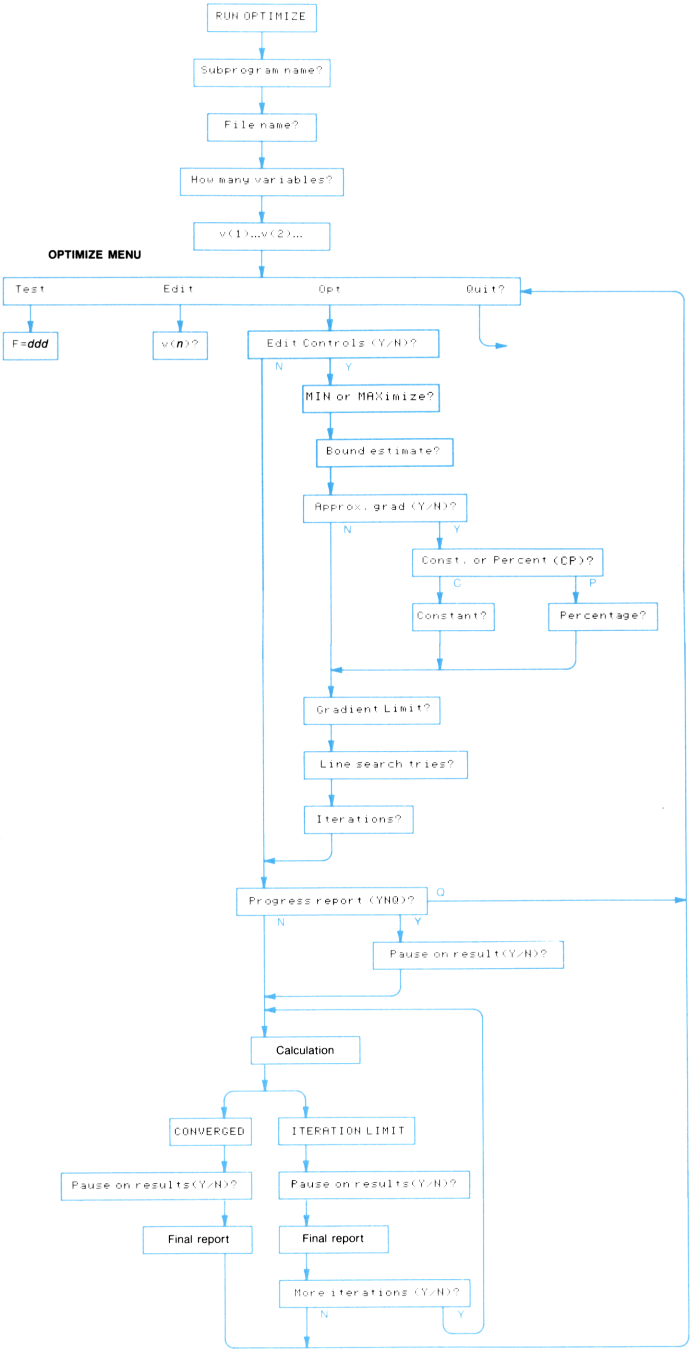
Overview

When you run the OPTIMIZE program to find local minima or local maxima, you go through a process similar to that for CFIT. You must:

1. Create a subprogram for the function you want to optimize.
2. Specify the name and location of the function subprogram.
3. Make an initial guess for the variables.
4. Test your guess (optional).
5. Optimize the function.

Flow Chart of OPTIMIZE Menus

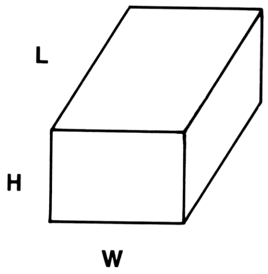
What follows is a flow chart of the steps involved in and the menus associated with the optimizing procedure.


RUN OPTIMIZE
Subprogram name?
File name?
How many variables?
v(1)...v(2)...
OPTIMIZE MENU
Test
Edit
Opt
Quit?
F=ddd
v(n)?
Edit Controls (Y/N)?
N
Y
MIN or MAXimize?
Bound estimate?
Approx. grad (Y/N)?
N
Y
Const. or Percent (CP)?
C
P
Constant?
Percentage?
Gradient Limit?
Line search tries?
Iterations?
Progress report (YNQ)?
Q
N
Y
Pause on result(Y/N)?
Calculation
CONVERGED
ITERATION LIMIT
Pause on results(Y/N)?
Pause on results(Y/N)?
Final report
Final report
More iterations (Y/N)?
N
Y

OPTIMIZE Example: A Big Box

Key in this OPTIMIZE example in the same way that you did for the CFIT example. In running this example, you will go through the five steps as outlined above.

Example: You are designing a box that will be used to mail widgets. You want the box to have dimensions that yield the largest volume while still being acceptable to your local carrier. The postal restrictions stipulate that the sum of the length and girth (perimeter of cross section) cannot exceed 100 centimeters. What is the maximum volume for your box and what are the dimensions?



Considering the postal restrictions and since you have maximum volume when the length and girth sum to 100, you can use these equations:

$$L + (2W + 2H) \leqslant 100$$

$$V = WHL$$

$$\begin{aligned} V(W,H) &= WH(100 - 2H - 2W) \\ &= 100WH - 2WH^2 - 2W^2H \end{aligned}$$

All dimensions must be greater than 0, so you can impose the additional constraints

$$0 < W + H < 50, \; W > 0, \; H > 0$$

First, type `EDIT EXAMPLES` and press [END LINE], and then enter the following subprogram called `BOX` and store it in a file named `EXAMPLES`. (For information on writing and storing subprograms, review section 12, "Subprograms and User-Defined Functions," in your *HP-71 Owner's Manual.*)

```
10 SUB BOX(P(),G(),V,Z)
20 W=P(1) @ H=P(2) @ BEEP
30 V=W*H*(100-2*W-2*H)
40 G(1)=2*H*(50-H-2*W)
50 G(2)=2*W*(50-W-2*H)
60 END SUB
```

Now enter the following keystrokes and see how easy `OPTIMIZE` is to use.

Input/Result

`RUN OPTIMIZE` [END LINE]

Run the program.

```
Subprogram name?■
```

The program prompts for the name of your subprogram.

`BOX` [END LINE]

```
File name?■
```

Prompts for the file where the subprogram is found.

`EXAMPLES` [END LINE]

```
How many variables?2
```

`OPTIMIZE` prompts for the number of variables in your function.

[END LINE]

Use the default of 2. Line 30 of the subprogram `BOX` shows that *W* and *H* are the two variables.

```
V(1)?0
```

The program prompts for an initial guess for the value of *W*. Line 20 of the subprogram identifies *W* as the first variable, `V(1)`, and *H* as the second variable, `V(2)`.

`5` [END LINE]

You don't know what the value is, but go ahead and guess 5.

```
V(2)?0
```

Prompts for a guess for *H*.

`6` [END LINE]

Guess 6 for *H*.

Input/Result

```
Test Edit Opt Quit?
```

After you enter the last guess, OPTIMIZE displays the Optimize menu.

[T]

Press [T] to evaluate $V(5,6)$. Evaluating the function at the current guess gives you the associated value of the function at that point.

```
F = 2340
```

[END LINE]

Return to the Optimize menu to prepare to enter the optimization routine.

```
Test Edit Opt Quit?
```

[O]

Select the optimization routine.

```
Edit Controls (Y/N)?
```

Asks if you want to edit the controls.

[Y]

Yes. You must edit the controls for this problem and all maximization problems because the program defaults to minimization.

```
MIN or MAXimize?MIN
```

Asks if you want to minimize or maximize (the default is to minimize).

MAX [END LINE]

```
Bound estimate?0
```

Prompts for the bound estimate.

3000 [END LINE]

It's at least 2340, so guess 3000.

```
Approx. grad (Y/N)?
```

The program asks if you want to have the gradient approximated for you.

[N]

Lines 40 and 50 in your subprogram do this for you, so you don't need the program to do it.

Input/Result

```
Gradient limit?.001
```

END LINE

Use the default limit to stop iterating.

```
Line search tries?10
```

END LINE

Use the default.

```
Iterations?25
```

END LINE

Use the default to set the upper limit on the number of iterations.

```
Progress report (YNQ)?
```

The program asks if you want a progress report for each iteration (Y or N) or if you want to return to the Optimize menu (Q).

N

When you press Y or N the program starts iterating. You hear a beep every time the subprogram is accessed from the main program.

```
1  |GRD|: 3.22274 E+001
2  |GRD|: 1.31881 E+001
3  |GRD|: 1.61895 E-001
4  |GRD|: 1.11935 E-003
CONVERGED
```

At this point, you see the norm of the gradient for each iteration followed by the CONVERGED message, displayed when the program converges.

END LINE

Press END LINE to continue.

```
Pause on results(Y/N)?
```

The program asks if you want to pause between each result.

Y

Press Y so you can scroll through the results at a comfortable pace using END LINE, assuming you don't have a printer or display monitor attached. (If you do have a printer or video display attached, you won't see this prompt. Printed results for this problem are shown on page 61.)

Input/Result

```
Fval:   9.25925926E+003
|Grd|:   7.4622188E-004
V(1):  1.666667467E+001
V(2):  1.666666905E+001
G(1):-6.128767544E-004
G(2):-4.257102044E-004
Test Edit Opt Quit?
```

Scroll through the output by pressing [END LINE] after each result. `Fval` is the maximum volume, `|Grd|` is the norm of the gradient (used in the determination by the program to stop iterating), `V(1)` is the value of *W*, `V(2)` is the value of *H*, `G(1)` and `G(2)` are the partials of *V* with respect to *W* and *H*. After the last item is displayed, you return to the Optimize menu.

[Q]

Press [Q] to quit the program.

```
Done
```

When you take the results for *V*, *W* and *H*, and then solve for *L* in the equation $V = WHL$, you have all the dimensions for your box. Your desired box size is 16.6667 by 16.6667 by 33.3333 centimeters with a volume of 9259.259 cubic centimeters.

Section 2

Curve Fitting

Introduction

The `CFIT` program allows you to enter your data into an array, store the array in a DATA file, retrieve the array from the file, and edit the data. Then, you can fit the data to a curve, examine the results, and optionally choose another model for another fit. Many common fit models reside in built-in library files, making specification of your model as simple as providing the name of the subprogram and the name of the built-in library file in which it resides. (Refer to appendix E for information on the library files. If you find that you need to write the model subprogram yourself, refer to appendix G, "Creating Your Own Model or Function Subprogram.")

`CFIT` permits you to fit a model function (specified in a subprogram) to a set of data. The program uses the FP Method* to minimize the Chi Square function associated with your model and the data set (for details of the FP method, refer to appendix C, page 75). The number of independent variables (n) and the number of data points in your data set (m) is limited only by the amount of available memory in your system. The number of parameters (k) in your model is limited to 20.

The model function you specify is represented by $F = F(X(\,),P(\,))$ where $X(\,)$ is a vector of length n and $P(\,)$ is a vector of length k for which there are k unknown parameters. For example, the model function might be a simple third-order polynomial. In this case, you can represent the unknown parameters (coefficients) by a vector of length 4 and the vector of independent variables reduces to a simple scalar value (in this case there is only 1 independent variable).

Model: $P_1 + P_2X + P_3X^2 + P_4X^3$

```
F  =  F(X,P())
   =  P(1)  + P(2)*X  + P(3)*X^2  + P(4)*X^3
```

If you make 12 observations (collect 12 data points) for the above example, your data array will be 12 by 3 (with 3 being derived from $n + 2$), and the 4 parameters to be determined are the coefficients of the polynomial model.

* As mentioned in section 1, the method used is the "Fletcher-Powell Method," abbreviated here and throughout the remainder of the manual as the "FP Method."

Running the Curve Fit Program

When you first run CFIT, by typing RUN CFIT END LINE, the main menu is displayed:

```
Data Edit Fit Quit?
```

The curve fit program is divided into three major parts. They are accessed from the main menu and are entitled Data, Edit, and Fit. The other option available in the main menu (Quit) allows you to quit the program.

> **Note:** If you exit the program by any means other than the Quit option, certain system functions are not maintained (for example, flag settings, rounding mode, and option base).

The Data part of the CFIT program contains commands for entering data points from the keyboard or from a file, saving entered data to a file, and printing the data array. The Edit part contains an editor for examining or modifying the data points. The Fit part contains the procedure for curve fitting. Each part is called from and returns to the main menu. The material that follows describes each of the three parts.

Working With Data

Data allows you to enter data points from either the keyboard or a data file. You can also save the data points in a file and, using the Print option, print the data points.

From the main menu you press D to display the Data menu:

```
Kbd Load Save Print?
```

All of the Data options are selected from this menu.

If you don't want to use any of the Data options, you can press Q to return to the main menu.

The Data Format

The data items consist of n independent variables, a dependent value, and a weight signifying the confidence level. These items are stored in an array of m data points (rows) and $n + 2$ columns.

For example, this 4 by 8 array represents a data set with 4 observations and 6 independent variables.

$$\begin{array}{cccccccc}
X_{1,1} & X_{1,2} & X_{1,3} & X_{1,4} & X_{1,5} & X_{1,6} & Y_{1,7} & W_{1,8} \\
X_{2,1} & X_{2,2} & X_{2,3} & X_{2,4} & X_{2,5} & X_{2,6} & Y_{2,7} & W_{2,8} \\
X_{3,1} & X_{3,2} & X_{3,3} & X_{3,4} & X_{3,5} & X_{3,6} & Y_{3,7} & W_{3,8} \\
X_{4,1} & X_{4,2} & X_{4,3} & X_{4,4} & X_{4,5} & X_{4,6} & Y_{4,7} & W_{4,8}
\end{array}$$

Giving Weights to Data

When CFIT creates a data array, each row has an associated column for the relative weight of that data. Data points given relatively small weights are considered "reliable," while those given relatively large weights are considered less reliable. When you give weights to data points, the value of the weights should be equal to the standard deviations of the dependent variables.

A reasonable approach to weighting is to use any reliable information you have regarding the true standard deviation as the weight. In the absence of any such information, use 1.

> **Note:** Do not use 0 as a weight. The Chi Square calculation involves a division by the weight. Using 0 results in a math error producing either /zero or 0/0 as an error message.

Entering Data From the Keyboard (Kbd)

If you want to enter a new set of data from the keyboard, follow the instructions below. This set of instructions, as well as other sets in this manual, follow a format that gives the step number, the display you will see, and the instructions on what to do to complete the step.

Step	Display	Instructions
1	Kbd Load Save Print?	Press K to start the process of creating a data set from the keyboard.
2	No. of indpt vars?■	Enter the number of independent variables. Two columns will be added to this number to form the internal array: one for the dependent variable y_i and one for the weighting factor w_i. *Any data already in an array will be destroyed when you create a new array.*
3	No. of data points?■	Enter the number of rows (data points) in the array.
4	X(1,1)?0	The program branches to the editor and displays the first element in the data array.

At this point the array has been created with all elements set to zero and the weights set to one. When you use the keyboard to enter the values of the elements, you simply edit their initial settings. The procedure for editing an element is shown in "Editing the Data," starting on page 31.

Loading Data From a File (Load)

If the data already exists in a file, you can load the data into an array by following the procedure below.

Step	Display	Instructions
1	Kbd Load Save Print?	Press L to load a data set from a file.
2	Clear data (Y/N)?	If you have data in the array, the Load option can append new data items to the existing set. If you wish to append the new data to the array, press N. To clear out the array, press Y. If no data is in the array when the Load option is selected, this question will not appear.
3	LOAD: File name?■	Enter the name of the data file. The file must be formatted as an *HPAF* file. For detailed information on the HPAF format, refer to appendix F. If a mass storage device is used, the file name must include the device specification (for example POINTS:TAPE).
4	Loading...	The program loads the file and then returns to the main menu.

Saving Data to a File (Save)

Once the data is entered, it is often a good idea to save it to a file for future use. Instructions on how to do this come next.

Step	Display	Instructions
1	Kbd Load Save Print?	Press S to save the current array of data in an HPAF file.
2	SAVE: File name?■	Enter the name of the file to write into. The file may reside either in RAM or on a mass storage device. If a mass storage device is used, the file name must include the device specifier (for example <POINTS:TAPE>). The program will create an HPAF file with the data in it.
3	Overwrite file (Y/N)?	If a file already exists with the name you supplied, the program will ask if you wish to overwrite it. If you press N, the program will again prompt for the file name. If you press Y, the program overwrites the previous file.
4	Saving...	The program saves the data and then returns to the main menu.

Printing the Data (Print)

Once the data is entered, you can also get a printed copy of the data array (assuming you have a printer attached). The procedure below explains how.

Step	Display	Instructions
1	`Kbd Load Save Print?`	Press [P] to print the data. The Curve Fitting Pac will send the data array to the current `PRINTER IS` device.
2	`Printing...`	While printing is in progress, this message is displayed.

If your HP-IL compatible printer is an 80-column printer, the output will be formatted as follows:

```
Row           X                 Y            W
  1:    5.000000E+000    3.480000E+001   8.500E-001
  2:    1.000000E+001    1.347000E+002   8.500E-001
  3:    1.400000E+001    1.593000E+002   8.500E-001
  4:    1.800000E+001    1.569000E+002   8.500E-001
  5:    2.400000E+001    1.322000E+002   8.500E-001
  6:    3.000000E+001    1.174000E+002   8.500E-001
  7:    3.500000E+001    1.325000E+002   8.500E-001
  8:    4.100000E+001    1.866000E+002   8.500E-001
  9:    5.000000E+001    3.422000E+002   8.500E-001
```

This example is a printout of the data used in the comprehensive `CFIT` example starting on page 40.

> **Note:** If your printer prints using a narrow field, like the HP 82162A Thermal Printer, the printout will be formatted in easily read columns.

After the data has been printed, the program returns to the main menu.

Editing the Data

`Edit` allows you to enter and modify arrays of data used by `CFIT`. This part of `CFIT` is selected from the main menu by pressing [E].

When you enter the array editor, the first display you see contains the value of the first element in the array:

`X(1,1)=`*nnn*

The display consists of three items:

- A letter giving the type of element displayed: `X` for an independent variable, `Y` for a dependent variable, and `W` for a weight.
- A pair of numbers in parentheses giving you the row and column address of the current element.
- The current value of the element.

Editing an Element

All of the HP-71 line editing features (such as >, <, I/R, BACK, -CHAR, and the command stack) are available in the array editor. If you are unfamiliar with how these features are used, refer to section 1, "Getting Started," in your *HP-71 Owner's Manual.*

Any time an element is visible it may be edited and the new value entered into the array. The element is edited by typing over the current value, and it is entered by pressing END LINE. All unassigned elements in a row are displayed as 0 except for the last one. The last element (the weighting for the observation) defaults to 1.

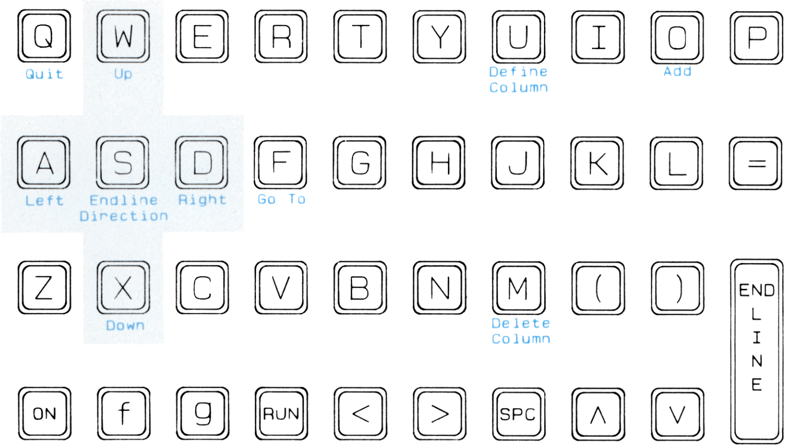
When you edit an element, you can use both numbers and numeric expressions for the value of an element. For instance, (1+SQR(25))/2 is just as acceptable as 3 for an entered value.

Moving Around the Array

When the editor is running, a number of keys have been redefined to help you move about within the array and to help you insert or delete rows and columns. These keys are broken into the following groups:

- The direction keys for moving through the array (W, D, X, and A).
- The command keys for manipulating columns and rows (U, O, M, and F).
- The endline direction keys (S in combination with the direction keys).
- The quit key (Q).

The following representation of the keyboard shows the keys that are redefined when you are in the array editor.



The Direction Keys. The direction keys are found in the cross on the keyboard diagram. You can use the direction keys in combination with the [f] key to move anywhere in the array. Once you get to the row and column you want, you can examine the contents of the element at that location and/or modify it.

The [W], [D], [X], and [A] keys are respectively the up, right, down, and left keys. These keys move you through the array one element at a time in the corresponding direction as shown in the following table.

Starting Element	Key	Direction	Destination
X(3,3)	[W]	up	X(2,3)
X(3,3)	[D]	right	X(3,4)
X(3,3)	[X]	down	X(4,3)
X(3,3)	[A]	left	X(3,2)

You can move to one of the boundaries of the array by pressing the [f] key and a direction key. You can think of the [f] as standing for the word "far."

- The [f] [D] and [f] [A] move to the far right and far left boundaries of the array.
- The [f] [W] and [f] [X] move far up and far down to the boundaries of the array.

All the direction keys will move across the boundaries to "wrap around" to the opposite side of the array. For example, in a 4 by 8 array, if you start at X(1,1) and press [f] [D], you will go to W(1,8) (the far-right side of the array). Then, if you press [D], you will see X(1,1) again.

> **Note:** If you are in the editor and want to enter an expression using letters that have been redefined, you must hold down the [g] key while typing the letters. For example, since the [F] key has been redefined as the GOTO command, you can only enter a weight as inf by holding down [g] and typing inf.

The Command Keys. The command keys can be associated with their gold, shifted functions on the HP-71 keyboard. The command keys and their functions are as follows:

- The [U] key selects the DEFine column command.
- The [O] key selects the ADD command.
- The [M] key selects the DELETE command.
- The [F] key selects the GOTO command.

The DEFine column command is used to assign values to the columns. It can be very useful when you want to enter data with a constant interval between points (for example, every year from 1954 to 1984 or 10-degree steps from 30 degrees to 120 degrees Fahrenheit). To do so:

1. Press U, at which time the program prompts

   ```
   Col, Start, Step?
   ```

2. Enter the column to fill, the starting value, and the step size, separated by commas. After the information has been entered, the program returns to regular editing.

Like all the keyboard commands, the define command can be used at any time while you are in the editor, regardless of where you are in the array. Also, if you get into this command accidently, press Q to return to regular editing.

The ADD command makes it easy to add a row or column to the array. To do so:

1. Press O, at which time the program prompts

   ```
   ADD: Row or Column?
   ```

2. Do one of the following:

   - Press R to add a row to the array; the program will display

     ```
     Add new row at?m+1
     ```

     with one more than the total number of rows (data points) as the default address for the new row.

   - Press C to add a column to the array; the program will display

     ```
     Add new column at?n+1
     ```

     with one more than the total number of independent variables as the default address for the new column.

   - Press Q to exit the command. The program will return to regular editing.

3. Enter the address of the new row or column. If the address coincides with an existing row or column, the array will open to create a space. The new row or column will be filled with default values.

If you add a row or column at the address of an existing row or column, the existing row or column address (and all those rows or column addresses beyond it) increases by one. For example, if you enter a new row at row 3, the old row 3 becomes row 4, row 4 becomes row 5, and so on.

The DELETE command is used to delete a row or column from the array. To do so:

1. Press M, at which time the program prompts

   ```
   DELETE: Row or Column?
   ```

2. Do one of the following:
   - Press [R] to delete a row from the array; the program will display

     ```
     Delete row number?n
     ```

     with the current row as the default.
   - Press [C] to delete a column from the array; the program will display

     ```
     Delete column number?n
     ```

     with the current column as the default.
   - Press [Q] to exit the command. The program will return to regular editing.
3. Enter the address of the row or column to delete. The program will display

   ```
   Delete row nn (Y/N)?
   ```

   or

   ```
   Delete col nn (Y/N)?
   ```

4. Press [Y] for *yes* to delete, or [N] for *no* to exit.

After the deletion, the program returns to regular editing at the last displayed element or, if the corresponding row or column or a row or column before it was deleted, to an element near the previously displayed element. If all rows or columns are deleted you will return to the main menu.

The GOTO command allows you to move directly to a specific element in the array. To do so:

1. Press [F], at which time the program prompts

   ```
   Row,Col?█
   ```

2. Enter the row and column address of the element in the array. After the address is entered, the program displays the the element for review or edit.

Endline Direction. After the [END LINE] key is pressed to enter an updated value into an array, the next array element is displayed for editing. The direction the program moves to display the next element is called the *endline direction*. The endline direction is set with the [S] key, the key in the middle of the direction keys.

The default direction is to the right, so that when [END LINE] is pressed the next element to the right is displayed. The first element in the next row is displayed after you enter the value in the right-most column. Using the default endline direction, you can easily input your data into a matrix by editing each element and pressing [END LINE].

There are three possible endline directions:

- To the right: This is set by pressing $\boxed{\text{S}}$ for set endline direction followed by the right direction key, $\boxed{\text{D}}$.
- Down the columns: This is set by pressing $\boxed{\text{S}}$ followed by the down direction key, $\boxed{\text{X}}$.
- No motion: This is set by pressing $\boxed{\text{S}}$ followed by $\boxed{\text{S}}$ again. This will cause the same element to be displayed after pressing $\boxed{\text{END LINE}}$.

When you press $\boxed{\text{S}}$, the program will display

```
Direction: D,X,S or Q?
```

At this point, you can press $\boxed{\text{D}}$, $\boxed{\text{X}}$, or $\boxed{\text{S}}$, depending on the endline direction you want. You can also press $\boxed{\text{Q}}$ if you want to escape the command. When you press one of these endline direction keys, the program sets the endline direction and returns you to the last element displayed. The following table summarizes the effects of setting the endline direction.

Direction Keys	Endline Direction	Current Element	Element After Pressing Endline
$\boxed{\text{S}}$ $\boxed{\text{D}}$	Right.	X(3,3)	X(3,4)
$\boxed{\text{S}}$ $\boxed{\text{X}}$	Down.	X(3,3)	X(4,3)
$\boxed{\text{S}}$ $\boxed{\text{S}}$	No motion.	X(3,3)	X(3,3)

Exiting the Editor. To exit (quit) the array editor press the $\boxed{\text{Q}}$ key. This will return you to the main menu. This, too, can be used regardless of where you are in the array.

Fitting the Curve

Once your data has been entered and, optionally, saved in a file, you are ready to fit the curve. The `Fit` procedure involves:

1. Specifying the model.
2. Editing the parameters.
3. Optionally evaluating Chi Square with respect to the current model parameters.
4. Optionally evaluating the model at one or more points.
5. Optionally editing the program controls.
6. Performing the fit.

Each of these steps will be described next.

Specifying the Model

The first step in the process is to specify the model subprogram to the CFIT program. This is done as shown below:

Step	Display	Instructions
1	Data Edit Fit Quit?	From the main menu, press [F] to begin curve fitting.
2	Subprogram name?■	Enter the model subprogram name to be called by CFIT.
3	File name?■	Enter the name of the file containing the subprogram.
4	No. of Model Parms?■	Enter the number of parameters in the model.

Every model, whether preprogrammed in the Curve Fitting Pac or written by you, has a given number of model parameters (k) and independent variables (n). Make sure that the number entered in step 4 agrees with the actual number in your model subprogram. If the number entered is too small, you will probably get an error when the model is evaluated. If the number is too large, you will get incorrect results.

Editing the Parameters

After you enter the number of parameters in your model, the program prompts for the value of the first parameter in the initial guess. You can edit each parameter as outlined below:

Step	Display	Instructions
1	P(1)?0	Enter the first parameter in the initial guess. The first time you run the program, the parameters default to 0.
2	P(2)?0	If there are more parameters to be entered, the program will prompt for them. Edit the remaining parameters as in step 1.
3	Csq Mdl Prms Fit Quit?	Once you have edited the last parameter, the program displays the Fit menu.

Options From the Fit Menu

Once you access the Fit menu, you can evaluate Chi Square at the current guess by pressing [C], evaluate the model by pressing [M], edit the parameters by pressing [P], fit the curve by pressing [F], or quit and return to the main menu by pressing [Q]. The first three of these choices ([C], [M], and [P]) are optional within the curve fitting process.

Evaluating Chi Square. If you press [C] to evaluate Chi Square the program will display

ChiSq=*nnn*

You then press [END LINE] to return to the Fit menu.

Evaluating the Model. To evaluate the model at any row (using current model parameters), follow the instructions detailed below.

Step	Display	Instructions
1	`Csq Mdl Prms Fit Quit?`	Press M to evaluate the model.
2	`Row # (or All)?■`	Enter the row to be evaluated, or enter A to evaluate the model at all points. After the program evaluates the model at the appropriate row(s), the results will be printed or displayed and the program will return to the Fit menu.

Editing the Program Controls. Before a curve is actually fit to your data, you can edit the program controls as shown below:

Step	Display	Instructions
1	`Csq Mdl Prms Fit Quit?`	Press F to start the procedure for fitting the curve.
2	`Edit controls (Y/N)?`	Press Y if you want to edit the controls that affect the numerical calculation, or N if you want to proceed with the calculation using the current controls. If you enter N at this point, the procedure continues at step 10.
3	`Min ChiSq estimate?0`	If you have reason to believe the minimum Chi Square exceeds a given positive value, enter the value here to improve program performance. The default estimate is zero.
4	`Approx. grad(Y/N)?`	If your model subprogram includes the gradient calculation (all of the models in the library do), press N and pick up the procedure at step 7. If it's necessary to approximate the gradient, press Y. (For information on the gradient and how it is used in this pac, refer to section 3, page 61.)
5	`Const. or Percent(CP)?`	Press P if you want to use a percentage of the parameters for Delta. If you want to use a specific constant for Delta, press C. (If you are unfamiliar with how Delta is used to approximate the gradient, refer to appendix C, page 82 for details.)
6a	`Constant?.00001`	Enter the constant for Delta, or use the default value.
6b	`Percentage?.001`	Enter the percentage for Delta, or use the default value.
7	`Gradient Limit?.001`	Enter the gradient limit, or use the default. This limit, compared with the norm of the gradient at each iterate, is the criteria used to determine convergence.

Step	Display	Instructions
8	`Line search tries?10`	Enter the maximum number of tries to be made in the line search routine. The default number of tries is 10. (For details on the line search routine, refer to appendix C, page 76.)
9	`Iterations?25`	Enter the maximum number of iterations to be made in the attempt to converge. The calculation will normally converge in less than 25 iterations, so, for convenience, the default is 25.

Performing the Fit

If you chose not to edit the program controls, or if you have completed editing them, you are ready to have the program perform the actual fit. `CFIT` makes its fit by finding the local minima of the Chi Square function. The "absolute best fit" may not be found. Instead, `CFIT` may converge to a "local best fit." (This situation is most likely to occur when you're using models containing periodic functions.) When the program prompts to determine if you want a progress report, it is ready to fit a curve to your data.

Step	Display	Instructions
10	`Progress report (YNQ)?`	At each iteration, Chi Square, the gradient norm, the parameters, and the gradient of Chi Square can be output. If this information is desired, press Y. This information is sent to the current `PRINTER IS` device. The default is to display the iterate number and the gradient norm on the current `DISPLAY IS` device. If you get to this point and decide that you don't want to go through with the calculation, press Q and the program will return to the Fit menu.

If you don't want a progress report, press N; the program will start iterating and you will see the display

```
00 |Grd|: n.nnnnnE nnn
```

For each iteration, the program will display the iterate number and the norm of the gradient.

If you do want a progress report (if you press Y) and don't have a printer attached, you will see the display

```
Pause on results(Y/N)?
```

If you press Y in response to this prompt, the program will stop between each result until END LINE is pressed. If you do not want to stop (if you press N), you will see the progress report at the current DELAY rate. Either way, the program starts iterating at this point and, if you have a printer attached, the progress report will be printed.

For an example of what the printed progress report looks like, refer to pages 47–49.

Getting the Results

Once the program has completed the required iterations, it will have converged or it will have reached the iteration limit without converging.

If the program did not converge, you will see the following:

Step	Display	Instructions
1	ITERATION LIMIT	Press END LINE to go to the next display.
2	More iterations(Y/N)?	Press Y to continue iterating (starting from the last iteration) with the same number of iterations as specified in the program controls, or press N to print the results.

Note: You may also see a numeric computation message, such as FIT ERR~~ Tries > Limit. If this happens, press f Cont to return to the Fit menu.

If the program did converge, you will see:

Step	Display	Instructions
1	CONVERGED	Press END LINE to continue.
2	Pause on results(Y/N)?	Press Y if you want to use END LINE to step through the results, or press N if you want the results output at the current DELAY rate. You will not see this prompt if you have a printer attached.

As with the progress report, the final results will be sent to the current PRINTER IS device. To see an example of printed final results, refer to page 49.

After the results have been printed, the program returns to the main menu.

A CFIT Example

The material that follows is a comprehensive CFIT example for you to key in. This example uses many of the data entry, editing, and function evaluation features available in this pac. It also uses POLY, one of the library subprograms available for CFIT. At the end, a discussion is included on interpreting the results.

There are eight main steps to go through in this example. You will:

1. Enter data into a working array.
2. Save this data into an HPAF DATA file.
3. Supply the name of the model subprogram from the library.
4. Supply control information.
5. Get the fit results.
6. Add another data point to the array for evaluation.
7. Evaluate the model at the extra point.
8. Interpret the results.

Setting Up the Problem

Example: Suppose you have taken the data in the following table.

No.	*X*	*Y*	*W*
1	5	34.8	.85
2	10	134.7	.85
3	14	159.3	.85
4	18	156.9	.85
5	24	132.2	.85
6	30	117.4	.85
7	35	132.5	.85
8	41	186.6	.85
9	50	342.2	.85

What are the coefficients of the fourth degree polynomial that best fit this data, and what is the value predicted by the model at $X = 27$?

Notice that the weights have all been given a value of .85. In this example you can assume that the standard deviation of the dependent variable (Y) is .85 and that it does not vary as a function of the independent variable (X).

Entering the Data

As the first step in this example, enter the data as follows:

Input/Result

`RUN CFIT` END LINE

Run the program.

```
Data Edit Fit Quit?
```

The main menu.

D

You need to enter the data...

```
Kbd Load Save Print?
```

K

from the keyboard.

```
No. of indpt vars?■
```

The program prompts for the number of independent variables in the problem.

`1` END LINE

There is only one dependent variable in this example, *X*.

```
No. of data points?■
```

The program prompts for the number of data points (or rows) in the table.

`9` END LINE

There nine rows in the table.

```
X(1,1)=0
```

The program asks for the value of the first element in the first row.

`5` END LINE

Enter the appropriate value for the element from the table.

```
Y(1,2)=0
```

Asks for the value of the second element in the first row.

`34.8` END LINE

```
W(1,3)=1
```

Asks for the value of the third element in the first row.

`.85` END LINE

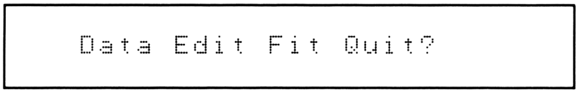
Enter the weight for row 1.

Input/Result	
X(2,1)=0	Asks for the value of the first element in the second row.
10 END LINE	Enter the first value in the second row of the previous table.
Y(2,2)=0	
134.7 END LINE	Enter the second value in the second row.
W(2,3)=1	
.85 END LINE	Enter the weight for row 2. Continue entering the data in the same manner for all the rows as the program prompts for the values.
⋮	
X(9,1)=0	The program asks for the value of the first element in the last row.
50 END LINE	
Y(9,2)=0	Asks for the second value in the last row.
342.2 END LINE	
W(9,3)=1	Asks for the weight for the last row.
.85 END LINE	
X(1,1)=5	The editor wraps around from the last element in the array to the first element in the array.
Q	Press the Q key to exit the editor and return to the main menu.

Saving the Data

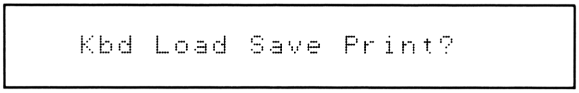
The second step in this problem involves saving the data to a file.

Input/Result



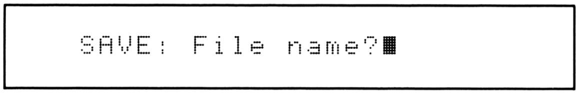
```
Data Edit Fit Quit?
```

D — Go to the Data menu.



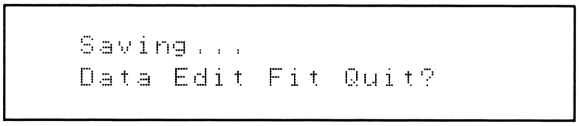
```
Kbd Load Save Print?
```

S — Save the data to a file named...



```
SAVE: File name?■
```

POLYDATA END LINE — POLYDATA.



```
Saving...
Data Edit Fit Quit?
```

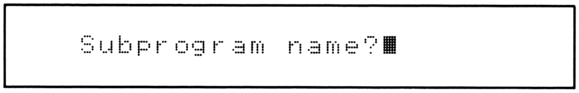
After saving the data to a file, the program returns to the main menu. (A printout of the data used in this example is shown on page 31.)

Specifying the Model Subprogram

The third thing to do is to specify the model subprogram to CFIT.

Input/Result

F — Start the curve fitting process.



```
Subprogram name?■
```

Asks for the name of the model subprogram.

POLY END LINE — Supply the name POLY since your polynomial function is covered by this library subprogram. (POLY is the built-in model subprogram that handles all polynomials through degree 19.)

Input/Result

```
File name?■
```

Asks for the file where the subprogram can be found.

FITLIB [END LINE]

POLY is found in the built-in library file FITLIB.

```
No. of Model Parms?■
```

Asks for the number of parameters.

5 [END LINE]

There are five coefficients in a fourth-degree polynomial.

```
P(1)?0
```

Asks for an initial guess for the first parameter. The program will go through all five parameters to allow you to supply an initial guess for each one.

[END LINE]

Enter the default value of 0 since you have no idea what the true value is. Do the same for the remaining parameters.

```
P(2)?0
```

[END LINE]

```
P(3)?0
```

[END LINE]

```
P(4)?0
```

[END LINE]

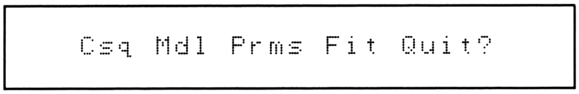
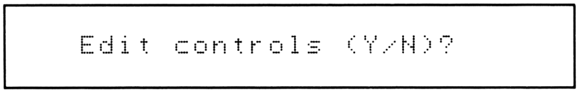
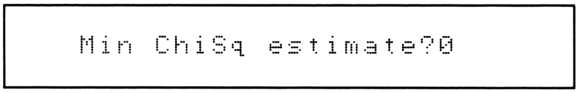
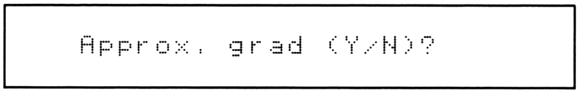
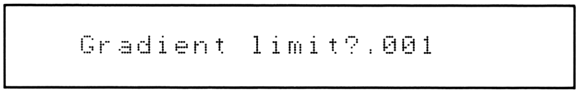
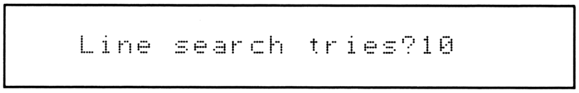
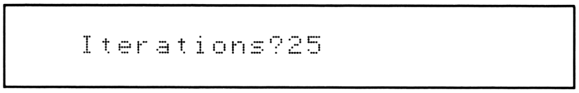
```
P(5)?0
```

[END LINE]

Editing the Controls

Now you need to go through the process of editing the program controls.

Input/Result

 Csq Mdl Prms Fit Quit?	The Fit menu.
F	Press F.
 Edit controls (Y/N)?	The program asks if you want to edit the controls.
Y	Yes. You need to edit the controls for this problem because the default gradient limit is unreasonable and, if used, eventually produces an error condition involving the gradient limit.
 Min ChiSq estimate?0	Asks for the minimum Chi Square estimate.
END LINE	You have no idea, so use the default value.
 Approx. grad (Y/N)?	Asks if you want the program to approximate the gradient.
N	No, because the subprogram calculates it.
 Gradient limit?.001	Asks for the gradient limit, which is the criteria used by the program to determine convergence.
1.00 END LINE	Enter 1.00 as the gradient limit. Do not use the default here; it is unrealistic for this problem.
 Line search tries?10	Asks for the limit on line search tries per iteration.
END LINE	Use the default.
 Iterations?25	Asks for the limit on the number of iterations.
END LINE	Use the default.

Input/Result

```
Progress report (YNQ)?
```

Asks if you want an iteration-by-iteration progress report showing intermediate results.

Getting the Results

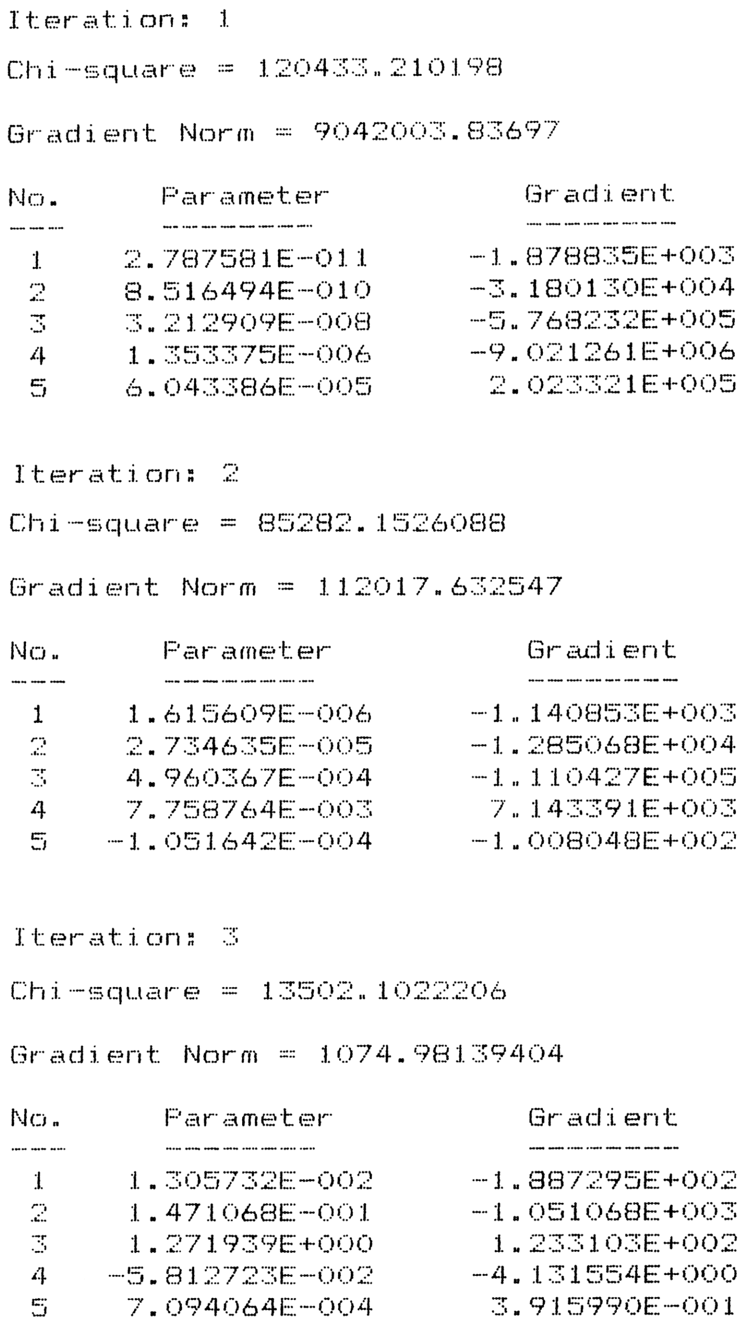
When you see the previous display, you are ready to get results.

Input/Result

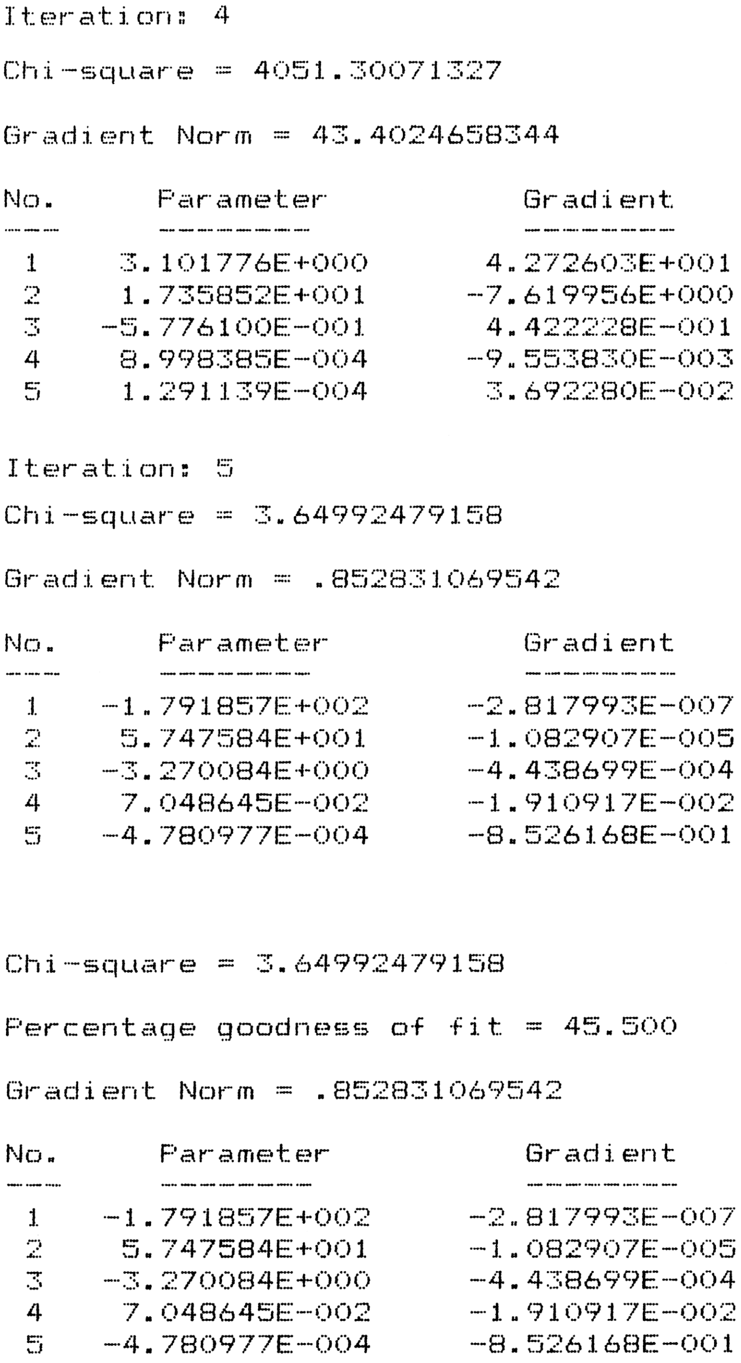
[Y]

Press [Y] for yes to see a progress report. After you press [Y], assuming you have a printer attached, the program prints the results. If you don't have a printer attached, the results will be displayed as in the example in section 1, starting on page 14. Assuming your printer is an 80-column printer, your intermediate and final results will be formatted as follows:

```
Initial values:

Chi-square = 373820.733564

Gradient Norm = 8383510901.56

No.       Parameter             Gradient
---       ---------             --------
 1      0.000000E+000       -3.866021E+003
 2      0.000000E+000       -1.181129E+005
 3      0.000000E+000       -4.455897E+006
 4      0.000000E+000       -1.876959E+008
 5      0.000000E+000       -8.381408E+009
```



```
Iteration:  1

Chi-square  =  120433.210198

Gradient  Norm  =  9042003.83697

No.        Parameter              Gradient
---        ---------              --------

 1      2.787581E-011        -1.878835E+003
 2      8.516494E-010        -3.180130E+004
 3      3.212909E-008        -5.768232E+005
 4      1.353375E-006        -9.021261E+006
 5      6.043386E-005         2.023321E+005


Iteration:  2

Chi-square  =  85282.1526088

Gradient  Norm  =  112017.632547

No.        Parameter              Gradient
---        ---------              --------

 1      1.615609E-006        -1.140853E+003
 2      2.734635E-005        -1.285068E+004
 3      4.960367E-004        -1.110427E+005
 4      7.758764E-003         7.143391E+003
 5     -1.051642E-004        -1.008048E+002


Iteration:  3

Chi-square  =  13502.1022206

Gradient  Norm  =  1074.98139404

No.        Parameter              Gradient
---        ---------              --------

 1      1.305732E-002        -1.887295E+002
 2      1.471068E-001        -1.051068E+003
 3      1.271939E+000         1.233103E+002
 4     -5.812723E-002        -4.131554E+000
 5      7.094064E-004         3.915990E-001
```



```
Iteration:  4

Chi-square  =  4051.30071327

Gradient  Norm  =  43.4024658344

No.        Parameter              Gradient
---        ---------              --------
 1      3.101776E+000         4.272603E+001
 2      1.735852E+001        -7.619956E+000
 3     -5.776100E-001         4.422228E-001
 4      8.998385E-004        -9.553830E-003
 5      1.291139E-004         3.692280E-002

Iteration:  5

Chi-square  =  3.64992479158

Gradient  Norm  =  .852831069542

No.        Parameter              Gradient
---        ---------              --------
 1     -1.791857E+002        -2.817993E-007
 2      5.747584E+001        -1.082907E-005
 3     -3.270084E+000        -4.438699E-004
 4      7.048645E-002        -1.910917E-002
 5     -4.780977E-004        -8.526168E-001


Chi-square  =  3.64992479158

Percentage  goodness  of  fit  =  45.500

Gradient  Norm  =  .852831069542

No.        Parameter              Gradient
---        ---------              --------
 1     -1.791857E+002        -2.817993E-007
 2      5.747584E+001        -1.082907E-005
 3     -3.270084E+000        -4.438699E-004
 4      7.048645E-002        -1.910917E-002
 5     -4.780977E-004        -8.526168E-001
```

For each iteration you are given

- I—the current iteration (guess) number.
- $P_1, \ldots, P_5$—the model parameters (or polynomial coefficients).
- $\partial\chi^2/\partial P_1, \ldots, \partial\chi^2/\partial P_5$—the gradient vector at $(P_1, P_2, \ldots, P_5)$.
- ChiSq—the value of Chi Square at $(P_1, P_2, \ldots, P_5)$.
- |Grd|—the gradient norm (measures the flatness of Chi Square).

Notice that after five iterations the Chi Square value was reduced from the initial value of 373,820.73 to 3.64992479158 (where convergence occurred). The final coefficients make the fourth degree polynomial look like this:

$$F(X) = P_1 + P_2X + P_3X^2 + P_4X^3 + P_5X^4$$

where

$P_1 = -1.791857\text{ E}2$

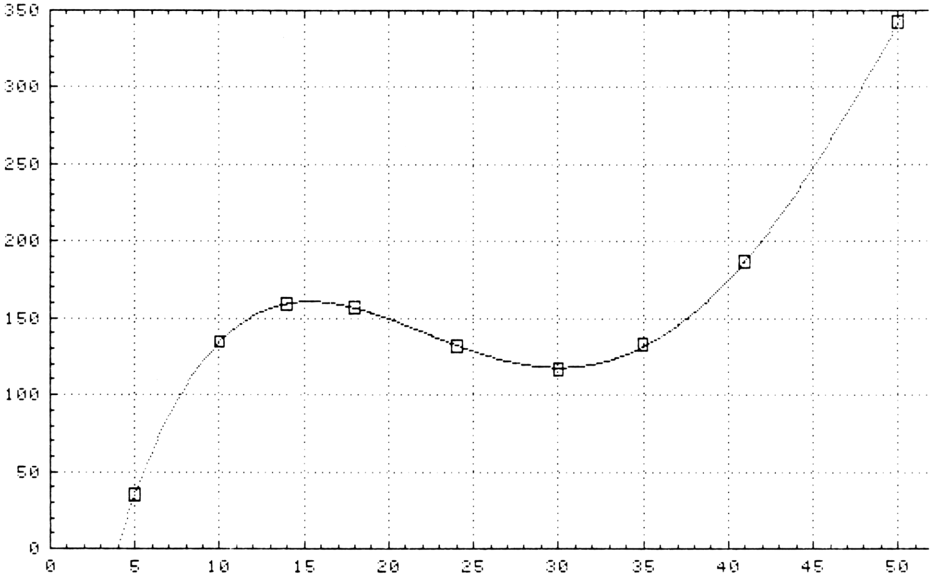
$P_2 = 5.747584\text{ E}1$

$P_3 = -3.270084\text{ E}0$

$P_4 = 7.048645\text{ E}-2$

$P_5 = -4.780977\text{ E}-4$

The following graph shows the nine data points and the polynomial function determined above.

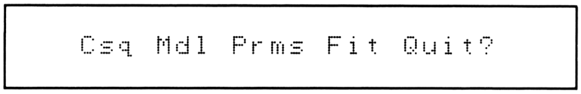


With the results, you also get something called *percentage goodness of fit*, which has the value 45.50. This number can be used in the interpretation of the results, which is discussed after the model evaluation.

Evaluating the Model

Now that you have the results of the fit, continue on with the example by adding a row and then evaluating the model at $X = 27$ to answer the original question: "What is the value predicted by the model at $X = 27$?"

Input/Result

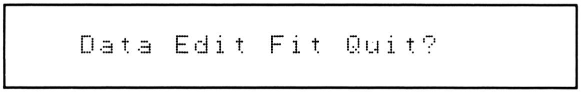


```
Csq Mdl Prms Fit Quit?
```

After printing the results, the program returns to the Fit menu.

Q

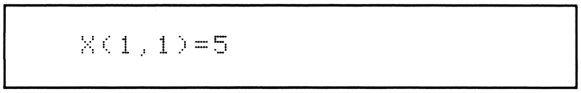
Press Q to return to the main menu.



```
Data Edit Fit Quit?
```

E

Press E to access the data editor.

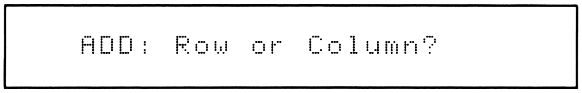


```
X(1,1)=5
```

The program enters the array at the first element.

O

The letter O has been redefined as the ADD command. (Refer to page 32 for a description of the redefined keys.)

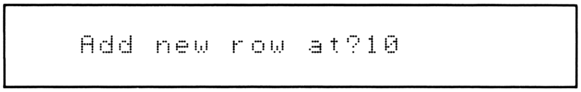


```
ADD: Row or Column?
```

Asks if you want to add a row or column.

R

Add a row.



```
Add new row at?10
```

Asks where to add the new row. The program defaults to one more than the current number of rows.

END LINE

Use the default to make the new row number 10.

Input/Result

```
Working...
X(10,1)=0
```

A new row is added to the array with default values.

27 [END LINE]

$X = 27$.

```
Y(10,2)=0
```

[END LINE]

Use the default value since this number is not used in the evaluation.

```
W(10,3)=1
```

The default weight.

[g] inf [END LINE]

While holding the [g] key down, type `inf`. This gives this row a weight of infinity. With this weight, the row will not be considered in the curve fitting process.

```
X(1,1)=5
```

[Q]

Exit the editor and return to the main menu.

```
Data Edit Fit Quit?
```

[F]

```
Subprogram name?POLY
File name?FITLIB
No. of Model Parms?5
P(1)?-179.185666734
P(2)? 57.4758354764
P(3)?-3.27008374086
P(4)? .070486453059
P(5)?-4.78097703425E-4
```

Access the Fit menu by scrolling through the subprogram and parameter prompts using the [END LINE] key.

Input/Result

```
Csq Mdl Prms Fit Quit?
```

[M] Press [M] to start the process for evaluating the model.

```
Row # (or All)?■
```

The program asks for the row number to be evaluated (or if all rows are to be evaluated).

`10` [END LINE] Row 10.

```
10: F = 122.074977996
```

The function value at $X = 27$.

[END LINE] Press [END LINE] to return to the Fit menu.

```
Csq Mdl Prms Fit Quit?
```

[Q]

```
Data Edit Fit Quit?
```

[Q]

```
Done
```

Interpreting the Results

An important point about the program is that achieving convergence does *not* necessarily mean that the model you have chosen is appropriate. `CFIT` merely tries to come up with the best solution for the model you have chosen. You can see from the graph on page 50 that the model chosen for this example is a good one. However, in the general case, where graphs may be inappropriate, statistics, specifically the Chi Square value, can be used to determine the acceptability of the model.

Assuming the dependent variables are normally distributed with standard deviations equal to the weights, and assuming the model fits your data well, the value identified by the program as Chi Square will be $\chi^2(\nu)$ distributed. The degrees of freedom (the number of data points minus the number of parameters) is ν. (The mean value of a $\chi^2(\nu)$ distributed variable is ν.) In this example the Chi Square value is 3.6499 and there are 4 degrees of freedom (9 data points minus 5 parameters). In general, if the above assumptions are satisfied, you can expect Chi Square to be close to the mean value ν.

Assuming the weights are valid, an unreasonably large value for Chi Square probably means that the model selected is inappropriate for the data.

The determination of "unreasonable" is usually made beforehand based on how much risk you are willing to take of rejecting a model when it is truly appropriate. For example, assume you are willing to take a risk of 10% of rejecting a valid model based on the previous criteria. You would see from Chi Square tables that 90% of the time a $\chi^2(4)$ (Chi Square with 4 degrees of freedom) distributed variable will be less than 7.78. Consequently, you would reject the model if the computed value exceeded this. Since 3.6499 is less than 7.78, you cannot reject the model for this example on that basis. In other words, the model for this example is reasonable.

Because of the utility of the Chi Square statistic in evaluating the validity of the model, the BASIC subprogram PCENTCHI has been built into the Curve Fitting Pac. This program eliminates the need to refer to Chi Square tables. PCENTCHI accepts values for U, the upper limit, and V, the degrees of freedom, and returns P, the probability that a Chi Square distributed variable is less than U.

CFIT calls PCENTCHI, passing it U (the value identified by CFIT as Chi Square) and V (the number of data points minus the number of model parameters) and obtaining in return the probability, P. The value identified by CFIT as the percentage goodness of fit is $100 \times (1 - P)$. This is the percentage of time that a Chi Square distributed variable with V degrees of freedom would exceed the CFIT value. If the percentage goodness of fit is less than your acceptable risk percentage (in this example, 10%), you would reject the model. Since the 45.50 returned by the program in this example is greater than 10, you have no reason to reject the model at the 10% significance level.

Section 3

Optimizing a Function

Introduction

Optimization is a term used to describe a class of problems in which the objective is to find the minimum or maximum value of a specified function. Often, the interest is focused on the behavior of the function in a particular region. Thus the goal becomes one of finding a *local* minimum or maximum. The OPTIMIZE program uses the FP Method to determine local minima or maxima for real-valued functions whose gradient vectors can be defined analytically at each point. The functions can have up to 20 variables.

Remember, the process you go through in OPTIMIZE involves:

1. Creating a subprogram for the function you want to optimize.
2. Running OPTIMIZE and specifying the name and location of your function subprogram to OPTIMIZE.
3. Making an initial guess for the variables.
4. Optionally:
   - Evaluating your function at the current variables.
   - Editing the current variables.
   - Editing the program control values.
5. Optimizing the function.

Creating the Function Subprogram

Before you run the program, you need to create a subprogram for your function and have it in memory. For information on creating the required subprogram, refer to appendix G, "Creating Your Own Model or Function Subprogram."

Running the OPTIMIZE Program

OPTIMIZE may be invoked through either a RUN command or a CALL command from another program.

You run OPTIMIZE by typing RUN OPTIMIZE [END LINE].

Specifying the Subprogram

Specifying the name and location of your function subprogram is the first step in OPTIMIZE. The set of instructions that follow, as well as other sets in this section, are formatted to give the step number, the display you will see, and the instructions on what to do to complete the step.

Step	Display	Instructions
1	Subprogram name?■	When you run OPTIMIZE, this is the first display you see. Enter the name of the function subprogram to be called.
2	File name?■	Enter the name of the file containing the subprogram.
3	How many variables?2	Enter the number of independent variables (the default is 2).

Make sure that your entry for the number of variables agrees with the number of variables actually in your function subprogram. If the number entered is too small, you will probably get an error when the function is evaluated. If the number is too large, you will get incorrect results.

Editing the Variables

After you enter the number of variables, the program displays the first variable for editing. You can enter (or edit) the values by stepping through each variable. As in CFIT, all the features of line editing in the HP-71 are available for use. (For more information on the keys used in line editing, refer to "Keyboard Operation" in section 1 of your *HP-71 Owner's Manual.*)

Step	Display	Instructions
1	V(1)?0	Enter the value of the first variable in the initial guess. Notice that the variables in the first guess default to zero.
2	V(2)?0	Enter the second variable. The program will prompt for the remaining variables in a similar manner. When you enter the last variable, the program will display the Optimize menu.

Options From the Optimize Menu

This is the Optimize menu:

```
Test Edit Opt Quit?
```

From it you can evaluate (test) your function at the current guess, edit the variables again, optimize (minimize or maximize) the function, or quit the program.

Testing the Function

To test the function using the current variables, follow the steps outlined below.

Step	Display	Instructions
1	Test Edit Opt Quit?	Press T.
2	F=*nnn*	The program evaluates the function at the current variables and displays the result. Press END LINE to return to the Optimize menu.

Editing the Controls

Once you have specified your subprogram, entered your variable values, and optionally tested your function, you are ready to optimize. Before you actually start the optimization, though, you have the option of editing the controls that affect the numerical calculations. The steps to do this are shown next.

Step	Display	Instructions
1	Test Edit Opt Quit?	Press O.
2	Edit controls (Y/N)?	Press Y if you want to edit the controls that affect the numerical calculation, or N if you want to proceed with the calculation using the current controls. If you enter N at this point, the procedure continues at step 11.
3	MIN or MAXimize?MIN	Enter MIN if you want to minimize the function, or MAX if you want to maximize it. The default is to minimize it.
4	Bound estimate?0	Enter the bound estimate (lower bound for minimizing and upper bound for maximizing). A good estimate of the bound can help the program to converge sooner than it otherwise would. The default is to 0.
5	Approx. grad (Y/N)?	If your subprogram includes the gradient calculation, press N and pick up the procedure at step 8. If you want the gradient approximated automatically, press Y.
6	Const. or Percent(CP)?	Press P if you want to use a percentage of the parameters for Delta. If you want to use a specific constant for Delta, press C. (Refer to appendix C for details on how Delta is used to approximate the gradient.)
7a	Constant?.00001	Enter the constant for Delta, or use the default value.
7b	Percentage?.001	Enter the percentage for Delta, or use the default value.
8	Gradient limit?.001	Enter the gradient limit, or use the default. This limit, compared with the norm of the gradient at each iterate, is the criterion used to determine convergence.

Step	Display	Instructions
9	`Line search tries?10`	Enter the maximum number of tries to be made in the line search routine. The default number of tries is 10. (For details on the line search routine, refer to appendix D.)
10	`Iterations?25`	Enter the maximum number of iterations to be made in the attempt to converge. The calculation will normally converge in less than 25 iterations, so, for convenience, the default is 25.

Performing the Optimization

If you chose not to edit the program controls, or if you completed editing them, you are ready to have the program perform the actual optimization. Remember that `OPTIMIZE` searches for a *local* minimum (or maximum). The absolute minimum (or maximum) may not be found. This is especially true for functions which have many critical points.

When the program prompts to determine if you want a progress report, it is ready to optimize your function.

Step	Display	Instructions
11	`Progress report (YNQ)?`	At each iteration, the current value of the variables, the function value, the gradient of the function, and the norm of the gradient can be output. If this information is desired, press Y. This information is sent to the current `PRINTER IS` device. The default is to display the iterate number and the gradient norm on the current `DISPLAY IS` device. If you get to this point and decide that you don't want to go through with the calculation, press Q. The program will return to the Optimize menu.

If you don't want a progress report, press N; the program will start iterating and you will see the display

```
00 |Grd|: n.nnnnnE nnn
```

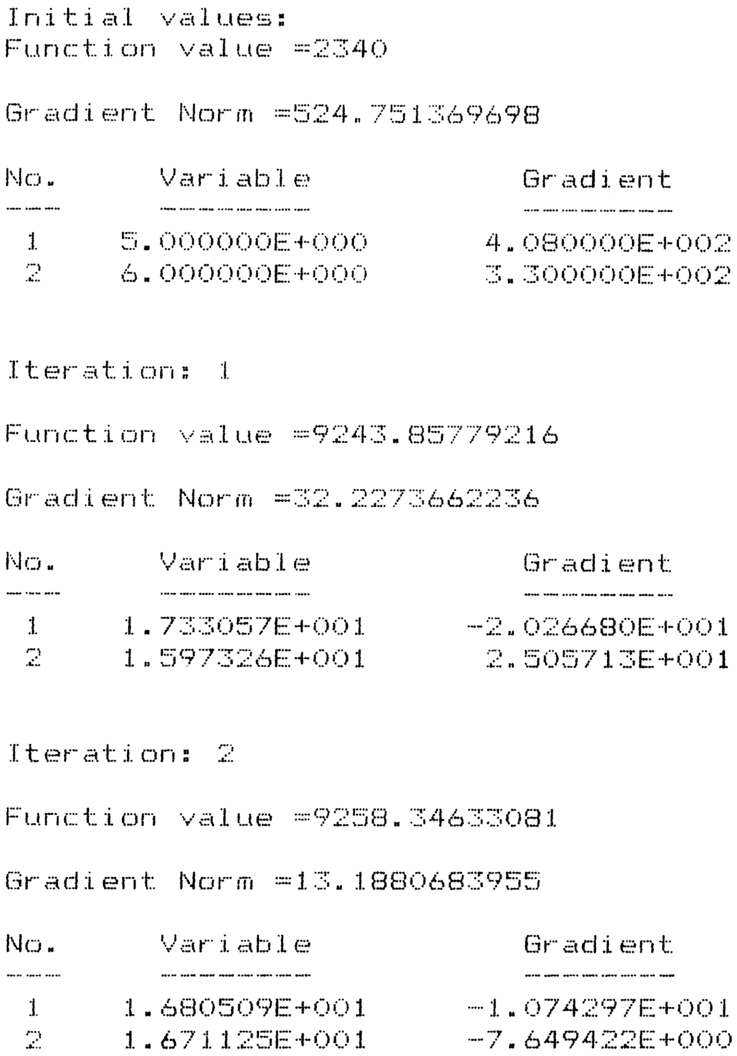
For each iteration, the program displays the iterate number and the norm of the gradient.

If you do want a progress report (if you press Y), and if you have no printer attached, you will see the display

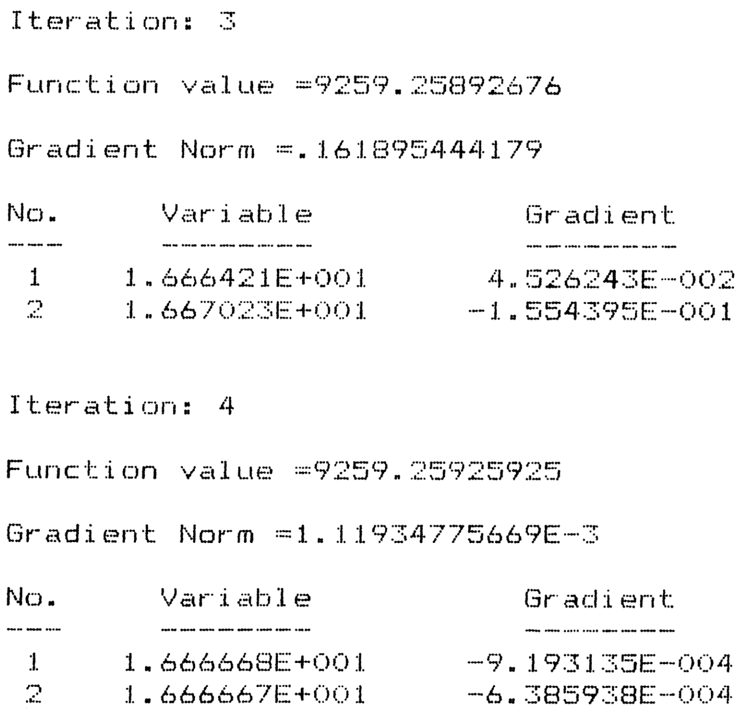
```
Pause on results(Y/N)?
```

If you press Y in response to this prompt, the program will stop between each result until END LINE is pressed. If you do not want to stop, press N to see the progress report at the current DELAY rate. Either way, the program starts iterating at this point and, if you have a printer attached, the progress report will be printed.

The report will look like this if you are using an 80-column printer:



```
Initial values:
Function value =2340

Gradient Norm =524.751369698

No.      Variable             Gradient
---      --------             --------
 1     5.000000E+000        4.080000E+002
 2     6.000000E+000        3.300000E+002


Iteration:  1

Function value =9243.85779216

Gradient Norm =32.2273662236

No.      Variable             Gradient
---      --------             --------
 1     1.733057E+001       -2.026680E+001
 2     1.597326E+001        2.505713E+001


Iteration:  2

Function value =9258.34633081

Gradient Norm =13.1880683955

No.      Variable             Gradient
---      --------             --------
 1     1.680509E+001       -1.074297E+001
 2     1.671125E+001       -7.649422E+000
```



```
Iteration: 3

Function value =9259.25892676

Gradient Norm =.161895444179

No.        Variable               Gradient
---        ----------             ----------
 1      1.666421E+001         4.526243E-002
 2      1.667023E+001        -1.554395E-001


Iteration: 4

Function value =9259.25925925

Gradient Norm =1.11934775669E-3

No.        Variable               Gradient
---        ----------             ----------
 1      1.666668E+001        -9.193135E-004
 2      1.666667E+001        -6.385938E-004
```

This example of a printed progress report is the progress report from “A Big Box,” the OPTIMIZE example in section 1, starting on page 22.

Getting the Results

Once the program has completed the required iterations, two situations can occur. Either the program will have converged or it will have reached the iteration limit without converging.

If the program did not converge, you will see the following:

Step	Display	Instructions
1	ITERATION LIMIT	Press END LINE to go to the next display.
2	More iterations(Y/N)?	Press Y to continue iterating (starting from the last iteration) with the same number of iterations as specified in the program controls, or press N to return to the optimize menu.

Note: You may also see a numeric computation message, such as FIT ERR~~ Tries > Limit. If this happens, press f Cont to return to the Optimize menu.

If the program did converge, you will see:

Step	Display	Instructions
1	`CONVERGED`	Press END LINE to continue.
2	`Pause on results(Y/N)?`	Press Y if you want to use END LINE to step through the results, or press N if you want the results output at the current `DELAY` rate. You will not see this prompt if you have a printer attached.

As with the progress report, the final results will be sent to the current `PRINTER IS` device. The report will be printed like this if you are using an 80-column printer:

```
Function value =9259.25925925

Gradient Norm =1.11934775669E-3

No.         Variable              Gradient

---         ---------             ---------

 1        1.666668E+001         -9.193135E-004
 2        1.666667E+001         -6.385938E-004
```

These printed results are the results from "`OPTIMIZE` Example: A Big Box" starting on page 22.

After the results are printed, the program will return to the Optimize menu.

A Word on Gradient

Since the gradient is an integral part of the optimizing process, a discussion of what the gradient is and how it is used in the FP Method is presented here.

For the function

$$F(X) = F(x_1, x_2, x_3, \ldots, x_k)$$

the gradient of F, denoted by ∇F, is defined by

$$\nabla F(X) = \begin{pmatrix} \partial F/\partial x_1 \\ \partial F/\partial x_2 \\ \partial F/\partial x_3 \\ \vdots \\ \partial F/\partial x_k \end{pmatrix}$$

Notice that the gradient of F is a vector whose length (number of elements) equals the number of variables. The components of $\nabla F(X)$ are the partial derivatives of F with respect to each coordinate. The negative of the gradient vector gives the direction of steepest descent (that is, the way in which X should be changed in order to cause the most rapid decrease in $F(X)$).

You might think that the most viable approach for obtaining the next estimate for the location of a minimum for F is to proceed some distance from the current estimate $X = (x_1, x_2, \ldots, x_k)$ in the direction indicated by the negative of $\nabla F(X)$. Indeed this technique (*Steepest Descent*) is in common use. However, this is not always a good strategy in that it can produce very slow convergence when the estimates get close to the desired location. The FP Method largely overcomes this difficulty by appropriately modifying the gradient vector to obtain a more productive search direction. For additional details of the FP Method, refer to "Fletcher-Powell Method" in appendix C.

Appendix A

Owner's Information

Limited One-Year Warranty

What We Will Do

The HP-71 Curve Fitting Pac is warranted by Hewlett-Packard against defects in materials and workmanship affecting electronic and mechanical performance, but not software content, for one year from the date of original purchase. If you sell your unit or give it as a gift, the warranty is transferred to the new owner and remains in effect for the original one-year period. During the warranty period, we will repair or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to a Hewlett-Packard service center.

What Is Not Covered

This warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than an authorized Hewlett-Packard service center.

No other express warranty is given. The repair or replacement of a product is your exclusive remedy. **ANY OTHER IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE-YEAR DURATION OF THIS WRITTEN WARRANTY**. Some states, provinces, or countries do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. **IN NO EVENT SHALL HEWLETT-PACKARD COMPANY BE LIABLE FOR CONSEQUENTIAL DAMAGES**. Some states, provinces, or countries do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights which vary from state to state, province to province, or country to country.

Warranty for Consumer Transactions in the United Kingdom

This warranty shall not apply to consumer transactions and shall not affect the statutory rights of a consumer. In relation to such transactions, the rights and obligations of Seller and Buyer shall be determined by statute.

Obligation to Make Changes

Products are sold on the basis of specifications applicable at the time of manufacture. Hewlett-Packard shall have no obligation to modify or update products once sold.

Warranty Information

If you have any questions concerning this warranty, please contact an authorized Hewlett-Packard dealer or a Hewlett-Packard sales and service office. Should you be unable to contact them, please contact:

- In the United States:

  Hewlett-Packard
  Personal Computer Group
  Customer Communications
  11000 Wolfe Road
  Cupertino, CA 95014

  Toll-Free Number: (800) FOR-HPPC (800 367-4772)

- In Europe:

  Hewlett-Packard S.A.
  150, route du Nant-d'Avril
  P.O. Box CH-1217 Meyrin 2
  Geneva
  Switzerland
  Telephone: (022) 83 81 11

  Note: Do *not* send units to this address for repair.

- In other countries:

  Hewlett-Packard Intercontinental
  3495 Deer Creek Rd.
  Palo Alto, California 94304
  U.S.A.
  Telephone: (415) 857-1501

  Note: Do *not* send units to this address for repair.

Service

Hewlett-Packard maintains service centers in most major countries throughout the world. You may have your unit repaired at a Hewlett-Packard service center any time it needs service, whether the unit is under warranty or not. There is a charge for repairs after the one-year warranty period.

Hewlett-Packard products are normally repaired and reshipped within five (5) working days of receipt at any service center. This is an average time and could vary depending upon the time of year and the work load at the service center. The total time you are without your unit will depend largely on the shipping time.

Obtaining Repair Service in the United States

The Hewlett-Packard United States Service Center for battery-powered computational products is located in Corvallis, Oregon:

Hewlett-Packard Company
Service Department

P.O. Box 999
Corvallis, Oregon 97339, U.S.A.

or

1030 N.E. Circle Blvd.
Corvallis, Oregon 97330, U.S.A.

Telephone: (503) 757-2000

Obtaining Repair Service in Europe

Service centers are maintained at the following locations. For countries not listed, contact the dealer where you purchased your unit.

AUSTRIA
HEWLETT-PACKARD Ges.m.b.H.
Kleinrechner-Service
Wagramerstrasse-Lieblgasse 1
A-1220 Wien (Vienna)
Telephone: (0222) 23 65 11

EASTERN EUROPE
Refer to the address listed under Austria.

GERMANY
HEWLETT-PACKARD GmbH
Kleinrechner-Service
Vertriebszentrale
Berner Strasse 117
Postfach 560 140
D-6000 Frankfurt 56
Telephone: (611) 50041

NORWAY
HEWLETT-PACKARD NORGE A/S
P.O. Box 34
Oesterndalen 18
N-1345 Oesteraas (Oslo)
Telephone: (2) 17 11 80

SWITZERLAND
HEWLETT-PACKARD (SCHWEIZ) AG
Kleinrechner-Service
Allmend 2
CH-8967 Widen
Telephone: (057) 31 21 11

BELGIUM
HEWLETT-PACKARD BELGIUM SA/NV
Woluwedal 100
B-1200 Brussels
Telephone: (02) 762 32 00

FINLAND
HEWLETT-PACKARD OY
Revontulentie 7
SF-02100 Espoo 10 (Helsinki)
Telephone: (90) 455 02 11

ITALY
HEWLETT-PACKARD ITALIANA S.P.A.
Casella postale 3645 (Milano)
Via G. Di Vittorio, 9
I-20063 Cernusco Sul Naviglio (Milan)
Telephone: (2) 90 36 91

SPAIN
HEWLETT-PACKARD ESPANOLA S.A.
Calle Jerez 3
E-Madrid 16
Telephone: (1) 458 2600

UNITED KINGDOM
HEWLETT-PACKARD Ltd
King Street Lane
GB-Winnersh, Wokingham
Berkshire RG11 5AR
Telephone: (0734) 784 774

DENMARK
HEWLETT-PACKARD A/S
Datavej 52
DK-3460 Birkerod (Copenhagen)
Telephone: (02) 81 66 40

FRANCE
HEWLETT-PACKARD FRANCE
Division Informatique Personnelle
S.A.V. Calculateurs de Poche
F-91947 Les Ulis Cedex
Telephone: (6) 907 78 25

NETHERLANDS
HEWLETT-PACKARD NEDERLAND B.V.
Van Heuven Goedhartlaan 121
NL-1181 KK Amstelveen (Amsterdam)
P.O. Box 667
Telephone: (020) 472021

SWEDEN
HEWLETT-PACKARD SVERIGE AB
Skalholtsgatan 9, Kista
Box 19
S-163 93 Spanga (Stockholm)
Telephone: (08) 750 2000

International Service Information

Not all Hewlett-Packard service centers offer service for all models of HP products. However, if you bought your product from an authorized Hewlett-Packard dealer, you can be sure that service is available in the country where you bought it.

If you happen to be outside of the country where you bought your unit, you can contact the local Hewlett-Packard service center to see if service is available for it. If service is unavailable, please ship the unit to the address listed above under Obtaining Repair Service in the United States. A list of service centers for other countries can be obtained by writing to that address.

All shipping, reimportation arrangements, and customs costs are your responsibility.

Service Repair Charge

There is a standard repair charge for out-of-warranty repairs. The repair charges include all labor and materials. In the United States, the full charge is subject to the customer's local sales tax. In European countries, the full charge is subject to Value Added Tax (VAT) and similar taxes wherever applicable. All such taxes will appear as separate items on invoiced amounts.

Computer products damaged by accident or misuse are not covered by the fixed repair charges. In these situations, repair charges will be individually determined based on time and materials.

Service Warranty

Any out-of-warranty repairs are warranted against defects in materials and workmanship for a period of 90 days from date of service.

Shipping Instructions

Should your unit require service, return it with the following items:

- A completed Service Card, including a description of the problem.
- A sales receipt or other proof of purchase date if the one-year warranty has not expired.

The product, the Service Card, a brief description of the problem, and (if required) the proof of purchase date should be packaged in adequate protective packaging to prevent in-transit damage. Such damage is not covered by the one-year limited warranty; Hewlett-Packard suggests that you insure the shipment to the service center. The packaged unit should be shipped to the nearest Hewlett-Packard designated collection point or service center. Contact your dealer for assistance. (If you are not in the country where you originally purchased the unit, refer to International Service Information above.)

Whether the unit is under warranty or not, it is your responsibility to pay shipping charges for delivery to the Hewlett-Packard service center.

After warranty repairs are completed, the service center returns the unit with postage prepaid. On out-of-warranty repairs in the United States and some other countries, the unit is returned C.O.D. (covering shipping costs and the service charge).

Further Information

Circuitry and designs are proprietary to Hewlett-Packard, and service manuals are not available to customers. Should other problems or questions arise regarding repairs, please call your nearest Hewlett-Packard service center.

When You Need Help

Hewlett-Packard is committed to providing after-sale support to all of its customers. To this end, our customer support department has established phone numbers that you can call if you have questions about this product.

Product Information. For information about Hewlett-Packard dealers, products, and prices, call:

(800) FOR-HPPC
(800 367-4772)

Technical Assistance. For technical assistance with your product, call the number below:

(503) 754-6666

For either product information or technical assistance, you can also write to:

Hewlett Packard
Personal Computer Group
Customer Communications
11000 Wolfe Road
Cupertino, CA 95014

Appendix B

Error and Status Messages

The Curve Fitting Pac programs return certain messages under specific conditions. Some of these are merely status messages, while others occur in response to an error. An incorrectly typed or constructed command will produce an error message. An error in a subprogram may produce an error message and halt execution of the program.

Several of the error messages are related to the amount of available memory. CFIT and OPTIMIZE contain tests for low memory conditions—conditions that would otherwise suspend program execution. If you encounter an error or warning that refers to low memory conditions, you should interrupt the program by pressing ATTN, catalog the memory files, and purge unneeded files to make more memory available. You can then press f CONT to proceed with the program.

BASIC Error Messages

The following is a list, in alphabetical order, of the error and status messages produced within the BASIC programs. Refer to page 117 for a list of the BASIC programs in this pac.

Message and Condition

Done
The program has ended. The HP-71 is now ready for the next task.

ERROR: 1 <= k <= 20
The number of unknowns must be between 1 and 20.

ERROR: N < 1
The number of independent variables and data points must be positive.

ERROR: Address
The given array element in a GOTO command does not exist.

ERROR: Array Too Large
There is not enough memory to hold the new data array (or enlarge the existing data array).

ERROR: Data Format
The file being read is not a properly formatted HPAF file.

Message and Condition
ERROR: File Not Found The file specified in a load operation or the file containing the named subprogram could not be found.
ERROR: HPIL An error was encountered when trying to print to an HP-IL device.
ERROR: Illegal ChiSq The given Chi Square estimate is invalid.
ERROR: Illegal Delta The given Delta is invalid.
ERROR: Iterations < 2 You must allow at least two iterations.
ERROR: Limit <= 0 The gradient limit must be positive.
ERROR: No Data There is no data. Data may be entered with the keyboard or load option.
ERROR: No Printer There is no printer on HP-IL.
ERROR: Nonexistent Col The given column in the ADD or DELETE command does not exist.
ERROR: Nonexistent Row The given row in the DELETE command does not exist.
ERROR: Not Enough Mem There is not enough memory to edit the data, or there is not enough memory to run the program.
ERROR: Not HPAF File The data file specified for a LOAD is not in the HPAF format.
ERROR: # Of Columns The file being added to the data array has a different number of columns than the array.
ERROR: PCENTCHI Failed The calculation of the percentage goodness of fit failed.
ERROR: SUB Not Found The subprogram could not be found.

Message and Condition

`ERROR: Tries < 2`
You must allow at least two line search tries.

`ERROR: You Need 1 Row`
The data array must contain at least one row.

`ERROR: You Need 3 Cols`
There must be at least three columns in the data array—allowing for one independent variable, the dependent variable, and the weight.

`ERROR dd SUBPROGRAM`
The subprogram cannot be called successfully.

`WRN: Deg. Freedom < 1`
The PCENTCHI calculation is meaningless for degrees of freedom less than 1.

Binary Error Messages

This next listing is the error messages that can be produced when an error is detected and reported from within one of the `FITLIB` binary subprograms that you can call. In addition to displaying the error or warning message, the subprogram passes back a *condition code* in the last variable that appears in the calling statement. The condition code will agree in magnitude with `ERRN`, but will additionally indicate whether the exceptional event was treated as a warning or error. A negative condition code indicates warning and a positive condition code indicates error. The occurrence of such a warning or error will *not* be detected by an `ON ERROR` statement and will *not* halt a running program. Consequently, it is usually best to test the condition code immediately after a call to any of the binary subprograms provided in this pac. `CFIT` and `OPTIMIZE` do this automatically for you.

If `CFIT` or `OPTIMIZE` do encounter an error, the programs pause and the **SUSP** annunciator appears. The program can be continued by pressing [f][CONT].

Note: Normal exits (including warnings and errors) will deallocate the scratch memory (system buffers) used by the binary subprograms making this memory available for other uses.

However, there are four exceptional events that are treated differently than the description above. If the subprogram is unable to assign a value to the condition code variable (errors 3015, 3016, and 3017), the error message is displayed as usual, but you will not be returned to the calling environment. Instead, the current file will be `FITLIB` and the environment will be that of the subprogram that detected the error. These errors are detected before any scratch memory is allocated. To recover, type `ENDSUB` from the keyboard to return to the calling environment.

A similar situation occurs if there is an attempt to display an error or warning and there is insufficient memory to do so. In this case the original error or warning message is replaced by the error message `ERR: Insufficient Memory`. Execution is halted with FITLIB as the current file and the local environment as that of the subprogram that attempted the error or warning message. In this last case, any scratch memory currently in use will *not* automatically be deallocated. You can recapture this memory by executing the keyword `KILLBUFF`, which explicitly deallocates all three of the system buffers that are utilized by this application.

These four exceptional events are, in normal use, extremely unlikely. In fact, the `OPTIMIZE` and `CFIT` programs protect you from the condition code errors by internal dimensioning and value checking. The insufficient memory error is not excluded but is unlikely since the error messages are short. If you are short of memory, it is far more likely that you will see the error `FIT ERR~~No Room` which indicates lack of memory to create required system buffers or to execute one of the many calls made by the binary subprograms to other subprograms.

Binary Error Listing

Errors and warnings detected by the binary subprograms are preceded by "`FIT`" with a "`~~`" appearing where the line number is normally displayed. The following table lists the error messages that can be produced within the binary subprograms.

Condition Code	Message and Condition
3001	`FIT ERR~~Aborted` The ATTN key was pressed during the execution of `FIT`, `FP`, or `CSQ`.
3002	`FIT ERR~~Pass By Value` A parameter that must be passed by reference to one of the binary subprograms has been passed by value.
3003	`FIT ERR~~Mat Not Sqr` The FP matrix must be a square matrix.
3004	`FIT ERR~~Too Many Vars` The number of unknowns (*k*) exceeds 20, or a matrix which should have length *k* is too large.
3005	`FIT ERR~~Bad Dimension` At least one of the arrays passed to a binary subprogram has the wrong length.
3006	`FIT ERR~~No Room` Insufficient memory to execute the binary subprogram.

Condition Code	Message and Condition
3007	`FIT ERR~~Complex Mat` An array passed to one of the binary subprograms is of COMPLEX type.
3008	`FIT ERR~~Complex Var` An argument to one of the binary subprograms is of COMPLEX type.
3009	`FIT ERR~~No Buffer` An expected system buffer was not found after return from a call.
3010	`FIT ERR~~Tries > Limit` The number of attempts within line search exceeded the supplied limit *with no* detectable improvement.
−3010	`FIT WRN~~Tries > Limit` The number of attempts within line search exceeded the supplied limit but *with* improvement in the current iterate.
3011	`FIT ERR~~Grad Delta=0` The parameter used to approximate the gradient in `GRADF` or `GRADM` is 0.
3012	`FIT ERR~~Int Type Var` An argument to one of the binary subprograms is of INTEGER type.
3013	`FIT ERR~~Int Type Mat` A matrix argument to one of the binary subprograms is of INTEGER type.
−3014	`FIT WRN~~Gradient=0` The matrix argument used to pass the current gradient is zero.
3015*	`FIT ERR~~CC-Int Type Var` The condition code variable is of INTEGER type.
3016*	`FIT ERR~~CC-Pass By Value` The condition code has been passed by value.
3017*	`FIT ERR~~CC-Complex Var` The condition code variable is of COMPLEX type.

* These values are not assigned to the condition code variable. However, `ERRN` will have these values.

Condition Code	Message and Condition
3018	FIT ERR~~NaN or Inf A NaN or Inf is encountered as a computed result or as an argument where disallowed.
−3019	FIT WRN~~Grad P+dP=P An attempt has been made to approximate the gradient with a value of Delta too small to effect a change in one of the parameters. In this case, at least one of the partials being approximated will erroneously be zero because of roundoff error.
3020	FIT ERR~~User A user function or model subprogram has set the condition code variable to 3020.
−3020	FIT WRN~~User A user function or model subprogram has set the condition code variable to −3020.

Condition Code Messages 3020 and −3020

The last two messages, 3020 and −3020, need a little more discussion to understand them completely.

These messages provide a mechanism for handling an error or warning detected within a model or function subprogram you write. If the value of the condition code variable is established as 3020, the error is displayed and the program halts just as with other error messages generated within the binary subprograms. (If the value is established as −3020, the warning is displayed and the program continues.)

If the condition code variable (C) is established as NaN, an error (message 3018) is generated. If the condition code is unchanged by the subprogram, its value upon exit remains zero and the calling program assumes normal processing has taken place and continues execution. Other nonzero values for the condition code variable assume an error if $C>0$, or warning if $C<0$, and that a message has already been displayed.

Appendix C

Numerical Methods

Fletcher-Powell Method

Both CFIT and OPTIMIZE use an algorithm commonly referred to in this manual as the Fletcher-Powell (FP) Method*. This algorithm accepts a function $F(X) = F(x_1, x_2, \ldots, x_k)$ with k variables and an initial guess $P = (p_1, p_2, \ldots, p_k)^T$ for the location of a local minimum, and then produces the next guess $P' = (p_1', p_2', \ldots, p_k')^T$. The manner in which this is accomplished is described next.

A k by k square matrix H is first initialized as the identity matrix. The use of the matrix H and the manner in which it is modified after each new guess distinguishes this method from other similar methods.

The negative gradient of F at the current iterate $(-\nabla F(P))$ is computed. This vector gives the direction of steepest descent at P. A unit search direction $S = (s_1, s_2, \ldots, s_k)^T$ is established via

$$S' = -H(\nabla F(P))$$

This vector is then normalized to the unit vector S via

$$S = S'/\|S'\|$$

Notice that initially H is the identity matrix, hence the direction for the first iterate is simply the direction of steepest descent. In theory it can be shown that the matrix H remains symmetric positive definite throughout the entire process and, consequently, that the function is always "decreasing" near P in the direction given by S.†

* R. Fletcher and M.J.D. Powell, "A Rapidly Convergent Descent Method for Minimization," *Computer Journal*, July 1963, pages 163-168.

† In practice, the modifications made to H can introduce roundoff errors, thus permitting H to lose these desirable properties and forcing a "restart" procedure to be employed in which H is reinitialized as the identity.

After a useful direction S from the current iterate P has been established, the next iterate is defined as $P' = P + t \times S$ for some appropriate choice of a positive scalar t. Thus P' lies on the ray emanating from P in the direction S.

The task that remains is the determination of an appropriate value for t. This task, commonly called *line search*, is not part of the FP Method. Various line search techniques are in common use. The one employed in this pac uses a modified cubic fit along the search ray. Details of the line search algorithm are given after the FP method is fully described. For now, observe that the value of t should be such that F is minimized near P in the direction S. This observation implies that line search is equivalent to minimization of a function of a single variable t. The function to be minimized is

$$h(t) = F(P + tS)$$

After an appropriate value of t has been obtained, the iterate P' is computed and the matrix H is updated as follows: (Notice that the denominators in the expressions for A and B are scalars.)

$$\begin{array}{lll} R = P' - P; & R = \Delta\, P & (k \text{ by } 1) \\ Q = G' - G; & Q = \Delta\, (\nabla F) & (k \text{ by } 1) \\ H = H + A - B & & \end{array}$$

The k by k matrices A & B are computed as follows:

$$A = (RR^T)/(R^TQ)$$
$$B = (HQQ^TH)\ /\ (Q^THQ)$$

Line Search

As previously mentioned, the object of the line search is the minimization of the function $h(t)$. Before launching into the explanation of the algorithm, you'll need to know some notational conventions. Notice again the function to be minimized:

$$h(t) = F(P + tS)$$

P is the current iterate and S is the search direction. The algorithm will selectively sample h at various values of t. Assuming selected values of t at $t = t_0$, t_1, and t_2, let us denote $h(t_0)$ by h_0, $h(t_1)$ by h_1 and $h(t_2)$ by h_2. Similarly, let us denote the slopes $h'(t_0)$, $h'(t_1)$, and $h'(t_2)$ by m_0, m_1, and m_2, respectively. Observe that given values t_0, t_1, and t_2, the determination of h_0, h_1, and h_2 can be obtained directly thru a CALL to the subprogram that represents the function F. The values of m_0, m_1, and m_2 are computed via:

$$m_i = (\nabla F(P + t_iS))^TS; \text{ for } i = 0,1,2$$

What we have on entry to the line search algorithm is P, the current iterate, S, the search direction (from the FP algorithm), and L, the estimated functional lower bound.

The first step in the algorithm is to compute m_0 and ensure that $m_0 \leqslant 0$. If this is not the case, the FP matrix H is reset to the identity and a new search direction S is determined. (This guarantees a new value of m_0 such that $m_0 \leqslant 0$ with equality only if the norm of the gradient is 0.)

Then, the algorithm initializes $t_0 = 0$ (corresponding to $P' = P$), and determines a value t_2 in such a manner that we expect the interval (t_0, t_2) to be a good search interval for the function h. More specifically, if $h_0 \leqslant L$ (this corresponds to a bad guess at L), L is first reset to $h_0 + \ (m_0/2)$. Notice that m_0 is less than 0; therefore, the new value of L is less than h_0.

$$t_2 = \text{MIN}\ [-2(h_0 - L)/m_0\ ,1]$$

This choice for t_2 deserves some explanation. The value given by $t' = -2(h_0 - L)/m_0$ is quite reasonable assuming that we are "close" to the minimum and L is a "good" guess at the minimum functional value. Near the minimum the second order terms in the Taylor expansion dominate, and h can be approximated by a quadratic. With these assumptions, t' is the location of the minimum of the quadratic that agrees with h (both value and derivative) at 0 and has minimum value L.

However, if the choice of L is not good, or if we are not close to the minimum, the search interval is restrained to maintain selective sampling near the current iterate by limiting the value of t_2. The value of 1 for this restraint is suggested by the literature and seems adequate in practice.

Now h_2 and m_2 are evaluated (the value of h and h' at t_2).

The four values that we now have, h_0, h_2, m_0, m_2, are sufficient to fit a cubic to h, determine the location of the minimum of the cubic and use this as a guess for t.

Roughly speaking that is what the algorithm does. However, in practice it is generally more productive to move at least one of the endpoints in the search interval (t_0, t_2) and try again, unless $m_2 > 0$ (corresponding to a sign change in the derivative of h). There are various cases to consider that depend on the sign of m_2 (and the value of h_2 relative to h_0 if $m_2 \leqslant 0$). These cases are illustrated by the following sketches and descriptions of the action of the algorithm in each case.

Case 1: $\mathbf{m_2} > \mathbf{0}$. This is the desirable case. In this case the algorithm performs the cubic fit obtaining a value t_1 that is an estimate of the location of the minimum of h. In the equation

$$Z = 3(h_0 - h_2)/(t_2 - t_0) + m_0 + m_2$$

the cubic degenerates to a quadratic if $2Z + m_0 + m_2 = 0$. In this case:

$$t_1 = t_0 + m_0(t_2 - t_0)/(2Z + 2m_0)$$

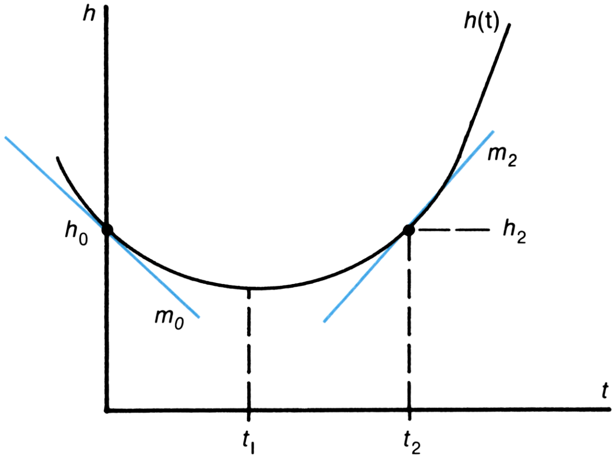
Otherwise

$$W = \sqrt{Z^2 - m_0 m_2}$$

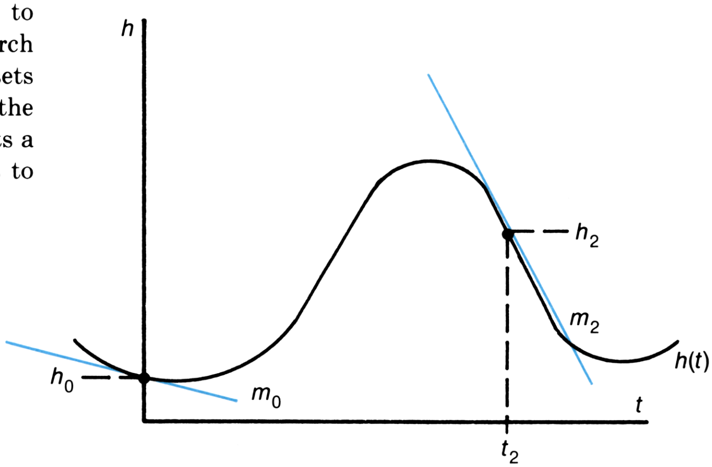
$$\mu = (m_2 + W - Z)/(2W + m_2 - m_0)$$

Also

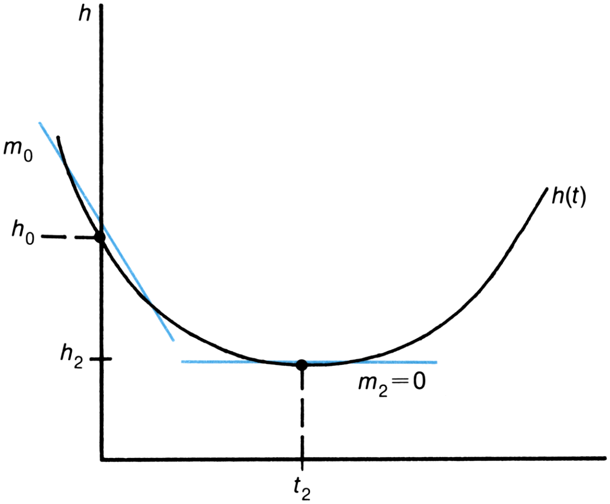
$$t_1 = t_2 - \mu(t_2 - t_0)$$



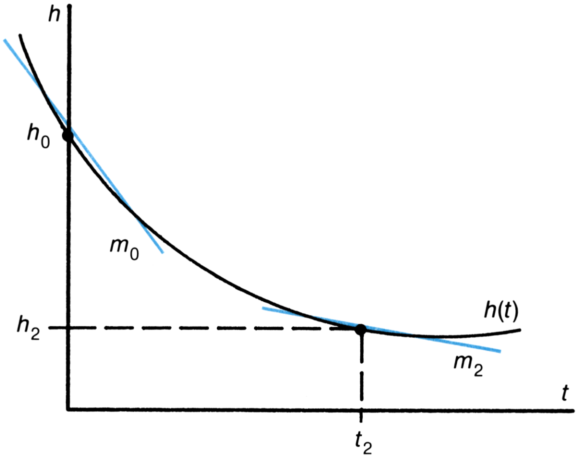
Case 2: $\mathbf{m_2} \leqslant \mathbf{0}$, $\mathbf{h_2} > \mathbf{h_0}$. In this case the algorithm changes the search interval to $(t_0, (t_0 + t_2)/2)$. This reduces the search interval by 1/2. Thus, the algorithms resets t_2 to $(t_0 + t_2)/2$ and repeats from the evaluation of h_2 and m_2. It also increments a counter that keeps track of the attempts to achieve case 1.



Case 3: $\mathbf{m_2 = 0, h_2 < h_0}$. In this case, the algorithm returns from the line search with $P' = P_2 = P + t_2S$



Case 4: $\mathbf{m_2 < 0, h_2 \leqslant h_0}$. In this case the algorithm changes the search interval to $(t_2, t_2 + 2(t_2 - t_0))$. This expands the search interval by a factor of 2. Thus the algorithm sets $t_0 = t_2$ and doubles the search interval size. It also, as in case 2, increments the counter that keeps track of the attempts to achieve case 1 and repeats from the evaluation of h_2 and m_2.



After an attempt at a cubic fit (from case 1), you have a quite reasonable choice t_1 for the location of the minimum for h.

If the fit was a success, t_1 is the location where the cubic polynomial, which agrees with h (in both value and derivative) at the search interval endpoints, achieves it's minimum value. If t_1 does not lie within the search interval, it is reset to the interval midpoint before the process continues $(t_1 = (t_0 + t_2)/2)$.

You have no guarantee at this point that t_1 is actually an improvement (h_1 is smaller) over the values at the endpoints of the search interval. Thus the next step is to evaluate h_1 and m_1.

If $h_1 < \text{MIN}(h_0, h_2)$, the algorithm returns from the line search with $P' = P + t_1 S$. Notice here that the algorithm does not spend time attempting to improve the value of t_1. In this type of application, determining the precise location of the minimum of h (especially for early iterations) is not nearly as important as minimizing the number of function evaluations per iteration. Thus, the line search algorithm attempts to produce an improved iterate and return quickly to the FP portion of the algorithm where a new search direction is selected.

If, however, h_1 is *not* less than $\text{MIN}(h_0, h_2)$, the algorithm does one of three things:

- If $m_1 < 0$, $h_1 \geqslant h_0$: It contracts the search interval to $(t_0, (t_0 + t_2)/2)$, increments the failure count, and repeats from the evaluation of h_2, m_2.
- If $m_1 < 0$, $h_1 < h_0$: It contracts the search interval to (t_1, t_2) (sets $t_0 = t_1$), increments the failure count, and retries the cubic fit.
- If $m_1 \geqslant 0$: It contracts the search interval to (t_0, t_1) (sets $t_2 = t_1$), increments the failure count, and retries the cubic fit.

Throughout the process, the failure count is getting incremented if progress is not being made. This value is checked against a user-supplied limit (TRIES) each time it is incremented. If it ever exceeds the limit, and the line search algorithm is unable to return with a new iterate that is "better" than the one on entry, a fatal error results. Here "better" means that a smaller (assuming the application is minimization) function value results (or the same function value with a smaller value for the magnitude of m).

Function Optimization

The FP Method and line search algorithms can be applied directly to the problem of optimizing a function. Although the FP Method is a minimization algorithm, it can also be used to maximize a function. This is due to the fact that local minima (maxima) for a function F are local maxima (minima) for the function $-F$.

Thus one approach to using the OPTIMIZE program to maximize a function would be to provide the negative of the desired function. This would require you to change the subprogram that encodes the function during searches for critical points.

MAX or MIN?

However, the user interface to the numerical algorithms is friendlier than that. You need only indicate that you want to find the maximum when prompted by the program. The interface will then set or clear the appropriate user flag that is examined at the time that F is evaluated. If appropriate, the value of F (and it's gradient) will be negated based on the value in this control flag. This control flag is user flag number 61.

Gradient

You can optionally indicate that you want the gradient of your function (or model) approximated. This is also implemented in the user interface through the use of a user flag that is examined at the time the function (or model) is evaluated. This flag is user flag number 62. The user interface responds to your preference by setting or clearing this flag.

However, you should be aware that the use of this friendly feature to approximate the gradient can make the program run significantly slower. Information on gradient approximation follows.

Gradient Approximation

If you are unable to compute the gradient in the subprogram that you use to specify your function (or model, if your application is curve fitting), you can have the gradient approximated for you. This can be very convenient, but has speed and accuracy considerations that you should be aware of.

Speed

Let's take an example using `CFIT`, assuming 20 data points and the model function given by

```
F = P1*SIN(P2*X) + P3*EXP(-X)
```

If the gradient is to be approximated, at least $k + 1$ calls (k = number of parameters) to the model function will be necessary to obtain the value for F and it's approximated gradient. In the above example, $k = 3$ and four calls are necessary. One call establishes the function value F. Three additional calls are necessary to evaluate F near $P(\ )$ in the three coordinate directions. For example, the approximation of the first coordinate in the gradient requires

$$\partial F/\partial P_1 = \frac{F((P_1 + \epsilon, P_2, P_3), X) - F}{\epsilon}$$

for some appropriately small non-zero value ϵ. Consequently, four calls to the model are required where only one would be required if the gradient were computed in closed form within the subprogram.

If this example is carried a bit further, the impact on speed can be seen more clearly. The function to be minimized is the Chi Square function that is associated with the model. The Chi Square function needs to call the model subprogram 20 times, each time obtaining both a model value and the gradient of the model to produce the Chi Square function value and it's gradient. If the gradient of F must be approximated, at least eighty (4 × 20) calls to the model subprogram are required to achieve each value of Chi Square and it's gradient. Examination of the line search algorithm shows that generally at least two functional evaluations will take place (and often more) for each iteration.

At the minimum then, for 20 data points, 160 calls will be made to the model function per iteration. This compares to 40 calls if the model gradient is supplied within the subprogram.

Most of the computation time will be spent within your BASIC subprogram. Consequently, anything you do within this subprogram to effect speed improvements (including the evaluation of the gradient in closed form) can have a significant impact on the total speed of the program.

Accuracy

One of the drawbacks to the approximation of the gradient is that the difference quotients can be a source of significant error. To illustrate, when the values in the numerator of the following equation are sufficiently close,

$$\partial F/\partial P_1 = \frac{F((P_1 + \epsilon, P_2, P_3), X) - F}{\epsilon}$$

their difference can produce 0 when the actual value can be quite large. The value entered in CFIT and OPTIMIZE as the constant or percentage is referred to here as Delta. If flag 63 is clear (Delta = constant), the value of ϵ is Delta. If flag 63 is set (Delta = percent), ϵ is the specified percentage of the parameter P as shown below:

$$\epsilon = \begin{cases} P \times (\text{Delta}/100) & \text{if } P \neq 0 \\ \text{Delta}/100 & \text{if } P = 0 \end{cases}$$

The main sources of error in the previous equation are the calculation of $P_1 + \epsilon$ and the difference that appears in the numerator. Both of these sources of error can require special handling to avoid inaccuracy.

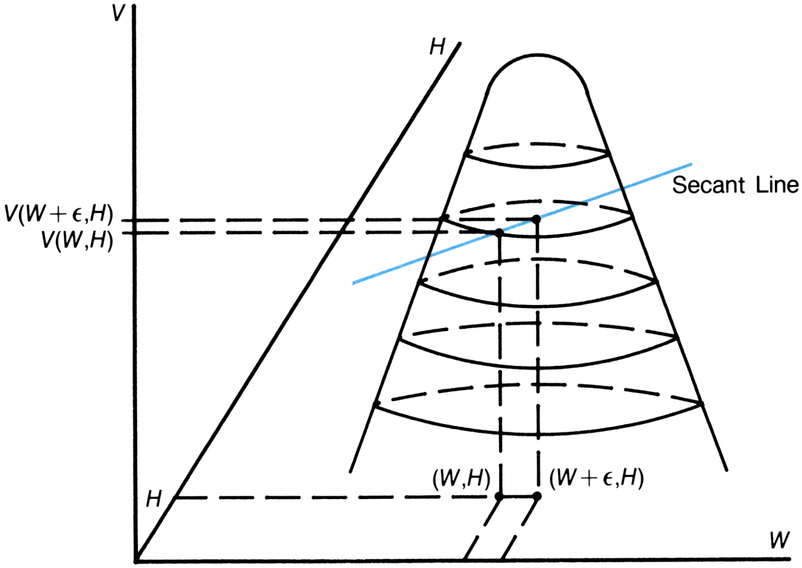
To illustrate these errors, consider the box example on page 22. The first line of the following table shows the result if $\nabla V(W,H)$ is computed at $W = H = 16.66664$ within the subprogram by a direct call to the given box subprogram. The value in line 1 is the exact value of the gradient vector and is useful for purposes of comparison with the results presented in the lines of the table representing gradient approximation (lines 2, 3, and 4). These last three lines were obtained by calls to the subprogram GRADF with flag 63 (relative approximation) set and with the indicated values for Delta.

	∇V(W,H) = (∂V/∂W,∂V/∂H)^T	
Delta (%)	**∂V/∂W**	**∂V/∂H**
(actual value)	2.66666240000 E−3	2.66666240000 E−3
0.001	−3.06000489601 E−3	−3.06000489601 E−3
0.0001	1.80000288000 E−3	1.80000288000 E−3
0.00001	0	0

This table illustrates some of the pitfalls that can beset the unwary user of gradient approximation. The previous equation shows the first coordinate ($\partial V/\partial W$) of $\nabla V(W,H)$ is computed as

$$\frac{\partial V}{\partial W} \approx \frac{V(W+\epsilon,H) - V(W,H)}{\epsilon}$$

where $\epsilon = W \times (D/100)$. In this case the basic problem can be traced to the fact that the point (W,H) is very close to the local maxima at $(W,H) = (50/3,50/3)$.



The thing to realize here is that the secant line joining the points $(W, H, V(W,H))$ to $(W + \epsilon, H, V(W + \epsilon, H))$ is being used to approximate the partial of V with respect to W. As in the two-dimensional case, where a secant is often used to approximate the derivative of a function, the choice of ϵ can become critical near local minima or maxima.

In fact, if ϵ is too large, the secant line can become a poor approximation and yield unsatisfactory results (as in the second and third lines of the table). If ϵ is too small, the function may be unable to distinguish between (W,H) and $(W + \epsilon,H)$ due to roundoff error in the functional evaluation. In this case, the difference quotient becomes zero (as in the fourth line of the table).

If possible, compute the gradient within your subprogram—especially if you need extremely accurate results.

Application to CFIT

The FP Method and line search algorithms described previously can be applied directly to a curve fit problem. This is true because there is a binary subprogram named CSQ in the file FITLIB that computes Chi Square (and it's gradient) of the user's model and data set. This subprogram, as well as others in the file FITLIB, can be called from the keyboard and called from within user-written programs. For information on CSQ and other user-accessible routines, refer to appendix D.

Minimizing Chi Square

CFIT attempts to fit your data by minimizing the Chi Square function associated with your data set and the specified model.

The Chi Square function associated with the model $F{=}F(X(\ ),P(\ ))$ at the current parameter iterate $P(\ )$ is defined by

$$\chi^2(P(\ )) = \sum_{i=1}^{m} [\,(F_i - Y_i)/W_i\,]^2$$

(1) Chi Square Function

Where F_i denotes $F = F(X(\ ),P(\ ))$ with $X(\ )$ the ith row in the Data Set. Y_i is the dependent variable and W_i is the weight.

$$\frac{\partial\chi^2(P(\ ))}{\partial P_j} = 2 \times \sum_{i=1}^{m} \frac{(F_i - Y_i)}{W_i^2} \times \frac{\partial F_i}{\partial P_j}$$

(2) Gradient Chi Square Function

With equal weights, equation (1) corresponds to the function that is minimized in the usual least squares method. Equation (2) provides the gradient of the Chi Square function in terms of the gradient of the model. Observe that there is no requirement that the model be linear. Indeed the model function may be quite general.

Difficult Cases

In addition to the difficulties associated with gradient approximation (refer to page 81 for information), there are areas where you may experience difficulty in obtaining the desired results. In some cases the difficulty can be avoided or its impact minimized. Three of these situations are described below. The information is not designed to provide a general solution to all the problems you might encounter, but to give you some ideas as to how to go about solving problems.

The Inability to Meet Convergence Criteria

The nature of the function being optimized (or the nature of Chi Square in CFIT) and the limits of machine precision can combine to keep the gradient norm from becoming sufficiently small to meet the convergence criteria used by OPTIMIZE and CFIT.

It is not unusual for the current iterate to be very close to the desired result, yet still have a reasonably large value for the norm of the gradient. This makes the selection of the limiting value used by CFIT and OPTIMIZE to detect convergence somewhat arbitrary. For this reason, you are given the ability in both CFIT and OPTIMIZE to change the value for the gradient limit by editing the program controls. A good strategy is to start with the gradient limit large, and, after convergence, reduce the limit and continue in an attempt to get closer to the desired result.

The error message

```
FIT ERR~~TRIES > LIMIT
```

often indicates that, because of machine precision and because of the nature of the function, the current iterate cannot be improved.

Sampling Outside the Intended Domain

An attempt to minimize the function $F(x,y) = (x-3)^2 + \sqrt{(x-y)}$ without special precautions will almost certainly result in sampling the subprogram for F at some point for which $x < y$. This, in turn, will result in the math error SQR(neg). The solution for this type of problem depends on the nature of the function. For this example, probably the most simple solution involves the realization that you can replace the $\sqrt{(x-y)}$ with $(x-y)^2$ without altering the solution.

Constrained Optimization

Some functions are subject to equality and inequality constraints. For example, a function to minimize such as $F(x,y) = (x-3)^2 + (y-2)^2$ can be subject to the constraint $G(x,y) = x + y - 4 = 0$ or, possibly, $x + y - 4 \leqslant 0$.

In simple cases like this the equality constraints can be used to solve for one variable in terms of the others. Substitution into the object function F will then eliminate one or more variables, and then the resulting function can be minimized.

Inequality constraints, such as $G(x,y) \leqslant 0$, are often handled by the use of a *penalty function*. One or more terms can be added to F to penalize samples taken from the wrong or undesirable region. For example:

$$\begin{aligned} H(x,y) &= F(x,y) + R/G(x,y) \\ &= (x-3)^2 + (y-2)^2 + R/(x+y-4) \end{aligned}$$

A minimum for H when $R = 1$ can be found using `OPTIMIZE`. By successively reducing R in small steps and by using the value obtained from the previous solution as an initial guess, you will obtain solutions which converge to the answer $(x,y) = (2.5, 1.5)$ where $F(x,y) = 0.5$. This type of procedure could even be "automated" by storing the updated value of R in a file that would be read by your subprogram for H.

Appendix D

User-Accessible Routines

This appendix describes the numerical subprograms that are available in the file FITLIB, their syntax, their calling relationships, and their memory requirements. Also, there is information describing the BASIC subprogram that produces the value of the Chi Square distribution function as well as information on two BASIC keywords, KEYWAIT$ and KILLBUFF, which reside in FITLEX.

Subprogram Description and Calling Syntax

What is described here is the required syntax in a CALL statement to the user-accessible subprograms. Of course, the actual variable names that you choose are immaterial. What is important is that the number, type and location of the arguments agree with the specifications that follow. In the case of the array parameters, the dimensions must be consistent with the problem. For example, the length (the number of elements) of the array *B*( ) that appears in the call to CSQ *must equal* the number of unknown parameters.

While none of the binary subprograms are base option dependent, CFIT and OPTIMIZE require base option 1.

The FP Subprogram

Functional Description. Performs one iteration of the FP method and the line search (described in appendix C) upon the function specified by the subprogram A1$ located in the file A2$.

```
CALL FP(A1$,A2$,B(),C(),D,E,F(,),G,H,I,Z) IN FITLIB
```

Inputs:

A1$ - Name of the subprogram encoding the function.
A2$ - Name of the file that contains the subprogram.
B() - Current iterate.
C() - Gradient at the current iterate.
D - Delta for gradient approximation (if flag 62 set).
E - Function value at the current iterate.
F(,) - FP matrix.
G - Line search iteration limit.
H - Estimate at functional bound.

Control Flags:

	61	62	63

Set
Clear
	MAX
	MIN
		Approx
		No Approx
			dP = %
			dP = Constant

Outputs:

- B() - New iterate.
- C() - Gradient at the new iterate.
- E - Function value at the new iterate.
- F(,) - Updated FP matrix.
- H - Updated bound estimate if original "bad."
- I - Norm of the gradient.
- Z - The condition code.

Comments: If the function has *k* variables, the arrays must have dimensions that produce the lengths shown at right.

Array	Length
B( )	k
C( )	k
F(,)	k by k

The FP subprogram samples the user's subprogram that encodes the function to be optimized by calls of the form:

```
CALL A1$(B(),C(),E,Z) IN A2$
```

The subprogram FP uses the results of such calls as described under "Fletcher-Powell Method," in appendix C.

The GRADF Subprogram

Functional Description: Produces an approximate gradient vector C() associated with the function specified by the subprogram A1$ that appears in the file A2$.

```
CALL GRADF(A1$,A2$,B(),C(),D,E,Z) IN FITLIB
```

Inputs:

- A1$ - Name of the subprogram encoding the function.
- A2$ - Name of the file that contains the subprogram.
- B() - Current iterate for the unknown parameters.
- D - Delta for gradient approximation.

Control Flags:

	61	62	63
Set	not used	not used	dP = %
Clear	not used	not used	dP = Constant

Outputs:

C( ) - Approximated gradient at B( ).
E - Function value at B( ).
Z - The condition code.

Comments: An error is returned if $D = 0$.

Array	Length
B( )	k
C( )	k

The jth coordinate, c_j, is evaluated by the formula:

$$c_j = \frac{F(b_1, b_2, \ldots, b_j + \epsilon, \ldots b_k) - F(b_1, b_2, \ldots, b_j, \ldots b_k)}{\epsilon}$$

F is the function whose gradient is being approximated. $\epsilon = D$ if flag 63 is clear. If flag 63 is set, $\epsilon = b_j(D/100)$ or $D/100$ if $b_j = 0$.

The GRADF subprogram is called by FP in the case where the gradient is to be approximated. The function subprogram must have the syntax that corresponds to the following call:

```
CALL A1$(B(),C(),E,Z) IN A2$
```

The function subprogram A1$ will be called $k + 1$ times to accomplish the stated objective.

The FIT Subprogram

Functional Description: Performs one iteration of the FP Method and the line search upon the Chi Square function associated with the specified model and data set.

```
CALL FIT(A1$,A2$,J(,),X(),B(),C(),D,E,F(,),G,H,I,Z) IN FITLIB
```

Inputs:

`A1$` - Name of the subprogram encoding the model.
`A2$` - Name of the file containing the model subprogram.
`J(,)` - Data array.
`X()` - Scratch array.
`B()` - Current iterate for the unknown parameters.
`C()` - Gradient at the current iterate.
`D` - Delta for gradient approximation (if `FLAG 62` set).
`E` - Chi Square value at the current iterate.
`F(,)` - FP matrix.
`G` - Line search iteration limit.
`H` - Estimate at functional bound.

Control Flags:

	61	62	63

Set
Clear
	MAX
	MIN
		Approx
		No Approx
			dP = %
			dP = Constant

Outputs:

`B()` - New iterate.
`C()` - Gradient at the new iterate.
`E` - Chi Square function value at the new iterate.
`F(,)` - Updated FP matrix.
`H` - Updated bound estimate if original "bad."
`I` - Norm of the gradient (usual norm).
`Z` - The condition code.

Comments: `FLAG 61` should be clear (set equal to 0) so that the Chi Square function is minimized. If the model has *k* parameters, *n* independent variables, and there are *m* data points, the arrays must have the lengths shown at right.

Array	Length
J(,)	m by (n + 2)
X( )	n + 2
B( )	k
C( )	k
F(,)	k by k

The `FIT` subprogram calls the `CSQ` subprogram which in turn makes calls to the model subprogram in order to achieve it's task—performing one iteration of the FP algorithm applied to the Chi Square function associated with the user's model. The calls made by `FIT` to `CSQ` have the form:

```
CALL CSQ(A1$,A2$,J(,),X(),B(),C(),D,E,Z) IN FITLIB
```

The subprogram `FIT` uses the results of such calls as described in "Fletcher-Powell Method," in appendix C.

The CSQ Subprogram

Functional Description: Evaluates the Chi Square function and its gradient associated with the specified model and data set at B().

```
CALL CSQ(A1$,A2$,J(,),X(),B(),C(),D,E,Z) IN FITLIB
```

Inputs:

A1$ - Name of the model subprogram.
A2$ - Name of the file that contains the model subprogram.
J(,) - Data array.
X() - Scratch array.
B() - Parameter array.
D - Delta for gradient approximation (if FLAG 62 set).

Control Flags:

	61	62	63

Set
Clear
	not used
	not used
		Approx
		No Approx
			dP = %
			dP = Constant

Outputs:

C() - The gradient of Chi Square at B().
E - The value of Chi Square at B().
Z - The condition code.

Comments: If the model has *k* parameters, *n* independent variables, and there are *m* data points, the arrays must have the lengths shown at right.

Array	Length
B( )	k
X( )	n + 2
J(,)	m by (n + 2)
C( )	k

The X() variable is used by CSQ to successively load in rows of the data array and execute calls to the user's model subprogram via:

```
CALL A1$(B(),X(),C(),E,Z) IN A2$
```

The results of each such call are used to update sums kept for the values of Chi Square and it's gradient.

The GRADM Subprogram

Functional Description: Produces an approximate gradient vector C() for the model function A1$ in file A2$.

```
CALL GRADM(A1$,A2$,X(),B(),C(),D,E,Z) IN FITLIB
```

Inputs:

A1$ - Name of the subprogram encoding the model.
A2$ - Name of the file that contains the model subprogram.
X() - Scratch Variable used to pass row data to model.
B() - Current Iterate for the unknown parameters.
D - Delta for gradient approximation.

Control Flags:

	61	62	63

Set
Clear
	not used
	not used
		not used
		not used
			dP = %
			dP = Constant

Outputs:

C() - Approximated gradient at B().
E - Model value at B().
Z - The condition code.

Comments: An error is returned if $D = 0$.

Array	Length
X( )	n + 2
B( )	k
C( )	k

The *j*th coordinate, c_j, is evaluated by the formula:

$$c_j = \frac{F(b_1, b_2, \ldots, b_j + \epsilon, \ldots b_k;\ x_1, x_2, \ldots, x_n) - F(b_1, b_2, \ldots, b_j, \ldots b_k;\ x_1, x_2, \ldots, x_n)}{\epsilon}$$

F is the model whose gradient is being approximated. $\epsilon = D$ if flag 63 is clear. If flag 63 is set, $\epsilon = b_j(D/100)$ or $D/100$ if $b_j = 0$.

The GRADM subprogram is called by CSQ in the case where the gradient of Chi Square is to be approximated. The model subprogram must have syntax corresponding to the CALL:

```
CALL A1$(B(),X(),C(),E,Z) IN A2$
```

The model subprogram A1$ will be called $k + 1$ times by GRADM to approximate the gradient of the model at a particular fixed row of the data array. The gradient of Chi Square is approximated by repeating this process m times (once for each data point or row in the array) and summing the results. A call to CSQ with the gradient of the model to be approximated will result in $m \times (k + 1)$ calls to the model subprogram.

The POLY Subprogram

Functional Description: Computes E, the polynomial value at x, and C(), the gradient with respect to the coefficients. The degree (up to 19) is determined by array sizes, and the evaluation proceeds by Horner's method.

```
CALL POLY(B(),X(),C(),E,Z) IN FITLIB
```

Where E and C() are returned as:

$$E = b_1 + b_2x + b_3x^2 + \ldots + b_kx^{k-1}$$
$$C(\,) = (1,\ x,\ x^2,\ \ldots,\ x^{k-1})$$

Inputs:

B() - Coefficients $(b_1, b_2, \ldots, b_k)$.
X() - (x, y, w).

Outputs:

C() - Gradient with respect to coefficients.
E - Polynomial value.
Z - The condition code.

Comments:

Array	Length
X( )	⩾ 1
B( )	k
C( )	k

The LIN Subprogram

Functional Description: Contains linear models of up to order 19.

```
CALL LIN(B(),X(),C(),E,Z) IN FITLIB
```

Where E and C() are returned as:

$$E = b_1x_1 + b_2x_2 + \ldots b_{(k-1)}x_{(k-1)} + b_k$$
$$C(\,) = (x_1,\ x_2,\ \ldots,\ x_{(k-1)},\ 1)$$

Inputs:

X() - Independent variables (k - 1 of them), dependent variable (Y), and weight (W).
B() - Linear coefficients (k of them).

Outputs:

C() - gradient.
E - Linear value E.
Z - The condition code.

Comments: The array X() is assumed to contain Y and W even though these values are not used by LIN.

Array	Length
X( )	k + 1
B( )	k
C( )	k

The PCENTCHI Subprogram

Functional Description: Evaluates the Chi Square distribution function, $P(x, v)$.

```
CALL PCENTCHI(X,V,P,Z)
```

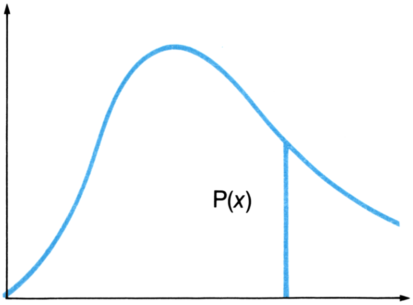
Inputs:

X - Value of Chi Square.
V - Degrees of freedom. (V must be a positive integer.)

Outputs:

P - Percentage of Chi Square distribution less than the given Chi Square value (for the given degrees of freedom). The value returned is rounded to the third decimal place.
Z - Error code (0 = okay, nonzero = error).

Comments: The Chi Square density function is really a family of curves—one for each positive degree of freedom. The cumulative distribution function is the area under the curve from zero to a given value of Chi Square.



If $x \leqslant 0$, then $P = 0$.

If $v \geqslant 40$, then the value of P is computed by using a quick approximation.

Let

$$x' = \frac{(\text{abs}(x)/v)^{1/3} - (1-2/(9v))}{\sqrt{(2/(9v))}}$$

Let ND be the normal distribution function. Its value at x can be approximated as follows:

$$p = .5(1 + d_1x + d_2x^2 + d_3x^3 + d_4x^4 + d_5x^5 + d_6x^6)^{-16}$$

Where

$d_1 = .0498673470$
$d_2 = .0211410061$
$d_3 = .0032776263$
$d_4 = .0000380036$
$d_5 = .0000488906$
$d_6 = .0000053830$

If $x > 0$, $\mathrm{ND}(x) = p$. If $x \leqslant 0$, $\mathrm{ND}(x) = 1 - p$.

Finally, $P(x, v)$ can be defined as:

$$P(x, v) = \mathrm{ND}(x')$$

This approximation is derived from the assumption that the Chi Square distribution for large degrees of freedom can be modeled by the normal distribution function with appropriate mapping of the domain.*

If $v < 40$ and $x > 80$, then $P(x, v) = 1$ which is accurate to the required three digits.

If $v < 40$ and $x \leqslant 80$, the value of P is computed using the following finite series:†

If v is odd:

$$P(x, v) = 2 \times \mathrm{ND}(\sqrt{x}\,) - 1 + \sqrt{(2/\pi)} \times \exp(\text{-}x/2) \times \sum_{i=1}^{(v\text{-}1)/2} \frac{(\sqrt{x}\,)^{(2i\text{-}1)}}{1 \times 3 \times 5 \ldots \times (2i - 1)}$$

Where $\mathrm{ND}(x)$ equals the left tail normal distribution computed according to the formula used previously.

If v is even:

$$P(x, v) = 1 - \exp(\text{-}x/2) \times \left\{ 1 + \sum_{i=1}^{(v\text{-}2)/2} \frac{(\sqrt{x}\,)^{2i}}{2 \times 4 \times 6 \ldots \times 2i} \right\}$$

* For information refer to the *Handbook of Mathematical Functions* by Abramowitz and Stegun, National Bureau of Standards, 1968, equations 26.2.19 and 26.4.14.

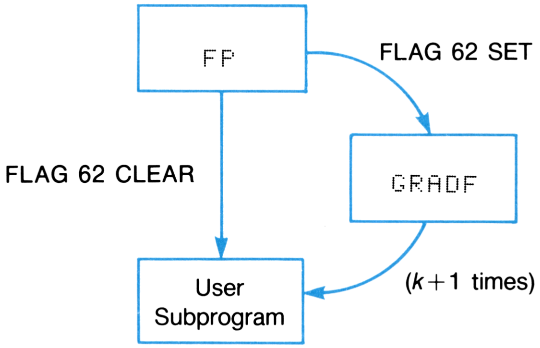
† Abramowitz and Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards, 1968, equations 26.4.4 and 26.4.5.

Calling Relationships

The subprograms in the file FITLIB (FP, FIT, CSQ, GRADF, GRADM, POLY, and LIN) are called from within CFIT and OPTIMIZE and also call each other. The following diagrams show the relationships between these subprograms. Arrows pointing down in the diagrams represent calls (multiple calls where indicated) to the lower level subprograms.

Subprograms Called by FP

Optimization



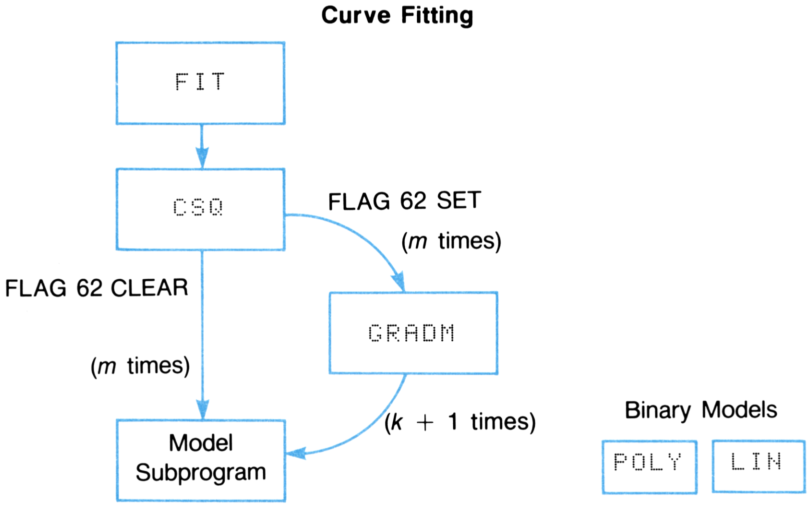
Flags 61, 62, and 63 are inputs acting as control flags, and k is the number of function variables.

The user subprogram computes the function value and the gradient of the target function. If the gradient is to be approximated, a call to GRADF replaces a call to the user subprogram. The syntax for a call to the user subprogram is as follows:

```
CALL A1$(B(),C(),E,Z,) IN A2$
```

The variables in the call are described in the discussion of FP, starting on page 87.

Subprograms Called by FIT



Flags 61, 62, and 63 are inputs acting as control flags, k is the number of unknown model parameters, and m is the number of data points.

The model subprogram (either user-written or provided in ROM) computes the model value and the gradient of the model with respect to the unknown parameters for a given data point. FIT functions in much the same way as FF, except that the target subprogram for minimization is not the user subprogram, but is CSQ. If the model gradient is to be approximated, calls to GRADM replace the calls to the model. The syntax for any model subprogram is identical to that for POLY and LIN.

Memory Requirements

There are seven binary subprograms in FITLIB. These subprograms are called by the programs CFIT and OPTIMIZE, but you can also call them from the keyboard or within your own program. What follows is information on the buffer and calling overhead associated with each routine and information on memory requirements for variable storage.

Buffer and Calling Overhead

Five of the seven binary subprograms create workspace buffers in main memory at execution time. The buffers are deallocated on exit, releasing this memory back to your system. Overhead memory is required by the operating system for each subprogram CALL. The buffer size (including header) and calling overhead are provided in the following table. The buffer size is a function of the number of unknowns (k), and the overhead is a function of the sum of the lengths (j) of the two string arguments in the call.

Buffer and Calling Overhead

Routine	Buffer	Buffer Size (Nibbles*)	Call Overhead (Nibbles*)	Total Bytes
FP	bFIT	$318 + 189k$	$301 + 2j$	$(619 + 189k + 2j)/2$
FIT	bFIT	$318 + 189k$	$339 + 2j$	$(657 + 189k + 2j)/2$
CSQ	bCHISQ	$108 + 21k$	$263 + 2j$	$(371 + 21k + 2j)/2$
GRADF	bGRAD	$95 + 21k$	$225 + 2j$	$(320 + 21k + 2j)/2$
GRADM	bGRAD	$54 + 21k$	$244 + 2j$	$(298 + 21k + 2j)/2$
POLY, LIN and other CFIT models	None	0	179	179/2
OPTIMIZE subprograms	None	0	160	160/2

* One nibble is equal to ½ byte of memory.

Notice that FP and FIT share the same buffer, as do GRADF and GRADM. Also, more than one buffer can be active at any given time. After a call to FIT, if the model gradient is to be approximated, all three buffers will be simultaneously active.

Variable Memory

In addition to the overhead associated with setting up a local environment within the subprogram, memory is required for the variables that appear as arguments in the calling statement. The amount of memory required for a numeric variable depends on the number of elements (n) in the variable ($n = 1$ for a simple numeric variable). The amount of memory required for a string variable depends on the length (l) of the string. The following table shows the memory requirements for variable storage.

Variable Storage

Type of Variable	Memory Required (Nibbles)
Real	$19 + 16(n)$
Short	$19 + 9(n)$
String	$23 + 2(l)$

As an example, consider a two-dimensional Real array that is 20 by 5 (has 100 elements). This array would require

$$19 + 16(100) = 1619 \text{ nibbles}$$

which is a little less than 810 bytes of memory.

Keywords

There are two keywords in FITLEX: KEYWAIT$ and KILLBUFF. To run CFIT and OPTIMIZE from memory (without the module plugged in), you must have several other files including FITLEX and FITLIB in memory as well.

KEYWAIT$

When the KEYWAIT$ function is executed, the HP-71 goes into a low power consumption state until a key is pressed and then returns the key name. This is similar to KEY$.

KILLBUFF

If a binary subprogram does not normally terminate when it encounters a low memory condition, the scratch memory allocated to it may not be deallocated. In this rare circumstance, the scratch memory can be reclaimed by executing KILLBUFF.

Appendix E

Library Subprograms

In order to provide easy specification of the more commonly used fit models to CFIT, a collection of subprograms that correspond to these models has been placed in built-in library files in this pac. The functions in the library files cannot be used in the OPTIMIZE program because OPTIMIZE and CFIT require different syntax (the number of parameters in the CALL to the subprograms differ).

If your model corresponds to one of the subprograms in one of the library files, you need not write a BASIC subprogram for it. When the program prompts you, you need only provide the name of the model subprogram that already exists. When you use the built-in models, remember that all models provided in this pac compute the model gradient as well as the model value.

> **Note:** The built-in models with trigonometric functions put your HP-71 into radians mode.

The file FITLIB contains two subprograms, POLY and LIN, that provide all the polynomial and linear models of one variable that CFIT can handle (a maximum of 20 model parameters). Appendix D, "User-Accessible Routines," contains information on POLY and LIN, starting on page 93. Also, there are 46 additional models in the file MODELS. These are described in the following table. In the model functions, *a* represents the first parameter ($P(1)$) in the equation, *b* the second ($P(2)$), and so on.

When you are using these models, don't forget that CFIT makes its fit by finding the local minima of the Chi Square function. The "absolute best fit" may not be found. Instead, CFIT may converge to a "local best fit." This situation is most likely when you're using models containing periodic functions.

Set of Models Supplied in Pac

No.	No. of Params	Model Name	Description	Model Function
1	2	ORPARAB	Parabola through origin.	$y = ax + bx^2$
2	1	ORLINE	Line through origin.	$y = ax$
3	2	HYPER	Hyperbola.	$y = a + b/x$
4	3	SHYPER	Second-order hyperbola.	$y = a + b/x + c/x^2$
5	2	POWER	Power.	$y = ax^b$
6	3	OPOWER	Offset power.	$y = a + bx^c$
7	4	POWER2	2 term power.	$y = ax^b + cx^d$
8	6	POWER3	3 term power.	$y = ax^b + cx^d + ex^f$
9	2	MPOWER	Modified power.	$y = ab^x$
10	3	OMPOWER	Offset modified power.	$y = a + bc^x$
11	2	RLINE	Reciprocal line.	$y = 1/(a + bx)$
12	3	LOGISTIC	Logistic.	$y = a/(1 + bc^x)$
13	2	ROOT	Root.	$y = ab^{(1/x)}$
14	2	SUPERGEO	Super geometric.	$y = ax^{(bx)}$
15	2	RTGEO	Root geometric.	$y = ax^{(b/x)}$
16	3	LINHYPER	Linear hyperbolic.	$y = ax + b + c/x$
17	2	RHYPER	Reciprocal hyperbola.	$y = x/(a + bx)$
18	2	EXP	Exponential.	$y = a \exp(bx)$
19	2	AEXP	Asymptotic exponential.	$y = a(1 - \exp(bx))$
20	3	DEXP	Double exponential.	$y = a (\exp(bx) - \exp(cx))$

Set of Models Supplied in Pac

No.	No. of Params	Model Name	Description	Model Function
21	2	RTEXP	Root exponential.	$y = a \exp(b/x)$
22	2	LINEXP	Linear exponential.	$y = ax \exp(bx)$
23	3	LOG	Logarithmic.	$y = a + b \ln(cx)$
24	3	RLOG	Reciprocal logarithmic.	$y = 1/(a + b \ln(cx))$
25	3	HOERL	Hoerl function.	$y = ab^x x^c$
26	3	RTHOERL	Root Hoerl function.	$y = ab^{(1/x)} x^c$
27	3	NORMD	Normal distribution.	$y = a \exp((x - b)^2/c)$
28	3	LOGNORMD	Log-normal distribution.	$y = a \exp((\ln(x) - b)^2/c)$
29	3	BETAD	Beta distribution.	$y = ax^b(1-x)^c$
30	3	GAMMAD	Gamma distribution.	$y = a(x/b)^c \exp(x/b)$
31	3	CAUCHYD	Cauchy distribution.	$y = 1/(a(x + b)^2 + c)$
32	3	SINU	Sinusoid.	$y = a \sin(b + cx)$
33	4	OSINU	Offset sinusoid.	$y = a \sin(b + cx) + d$
34	3	COSINU	Cosinusoid.	$y = a \cos(b + cx)$
35	4	OCOSINU	Offset cosinusoid.	$y = a \cos(b + cx) + d$

Set of Models Supplied in Pac

No.	No. of Params	Model Name	Description	Model Function
36	3	SINF2	2 term sine Fourier.	$y = b + c \sin(ax)$
37	4	SINF3	3 term sine Fourier.	$y = b + c \sin(ax) + d \sin(2ax)$
38	5	SINF4	4 term sine Fourier.	$y = b + c \sin(ax) + d \sin(2ax) + e \sin(3ax)$
39	3	COSF2	2 term cosine Fourier.	$y = b + c \cos(ax)$
40	4	COSF3	3 term cosine Fourier.	$y = b + c \cos(ax) + d \cos(2ax)$
41	5	COSF4	4 term cosine Fourier.	$y = b + c \cos(ax) + d \cos(2ax)$ $+ e \cos(3ax)$
42	4	FOURIER2	2 term Fourier.	$y = b/2 + c \cos(ax) + d \sin(ax)$
43	6	FOURIER3	3 term Fourier.	$y = b/2 + c \cos(ax) + d \sin(ax)$ $+ e \cos(2ax) + f \sin(2ax)$
44	8	FOURIER4	4 term Fourier.	$y = b/2 + c \cos(ax) + d \sin(ax)$ $+ e \cos(2ax) + f \sin(2ax)$ $+ g \cos(3ax) + h \sin(3ax)$
45	3	COSH	Hyperbolic cosine.	$y = a + \cosh(b + cx)$
46	3	SECH	Hyperbolic cosecant.	$y = a + \operatorname{sech}(b + cx)$

Suppose you suspect that the data you want to fit a curve to tends to a parabola through the origin. Rather than immediately setting out to create a model function for a parabola through the origin, you should look through the table above. The table shows that the model function has been written for you and is called ORPARAB in the file MODELS. So, you don't have to write any BASIC routines. Instead, when the curve fit program asks for the name of the model and the file, you type ORPARAB and MODELS, respectively.

Appendix F

Applications File Format (HPAF)

The HP-71 Curve Fitting Pac stores data in files according to a prescribed format. This format, the applications file format (*HPAF*), is intended to allow exchange of data between various programs. The format provides room for information that describes the structure of the data so that various programs may make use of and exchange the data.

HPAF files are of type DATA, and may reside in either the HP-71 memory or a mass storage device.

The HPAF files are composed of three major sections—the *header*, the *data records*, and an optional *descriptor block*. An example of such a file is described in the following table.

Record	Contents	Description
0	"HPAF*NNS*"	Type string: two numbers, one string.
1	4	There are four records of data.
2	12	The descriptor block starts at 12.
3	77,9.3,"RED"	First data record.
4	78,9.4,"BLUE"	Second data record.
5	81.5,10.3,"GREEN"	Third data record.
6	82.9,10.4,"GREEN"	Last data record.
.		Empty data records.
.		Empty data records.
12	"TEMP",1,"KELVIN" "COLNAMS",3,"TEMP" "VISCOSITY","COLOR"	Descriptor block.

Header Information

The header must contain the following items:

Record	Description
0	Record 0 contains a type string. The first four characters indicate the file is an HPAF file. The remaining characters describe the number of data items in each record and their type. For example, in "HPAF*NN*S" the characters "*NN*S" indicate that there are three items in each record (the first two are numbers and the third a string).
1	Record 1 contains the number of data records that contain information. This number can be less than the total number of available records (allowing room for additional records to be added later followed by the optional descriptor block).
2	2 contains the record number of the optional descriptor block. If no descriptor block is present, this number should be zero.

Data Records

The data records start at record 3 and must end before the descriptor block. Note that all data items for each record must fit within each logical record so that any record can be accessed randomly. To compute the optimal logical record length for the file, remember that each number written in the record occupies 8 bytes, and each string occupies 3 bytes plus the number of bytes in the string. For example, if each record is going to hold two numbers and a ten character string, the record length must be at least $2 \times 8 + 3 + 10$, or 29 bytes. For more information about creating data files, refer to "Data Files," in section 14 of the *HP-71 Owner's Manual.*

Descriptor Block

The descriptor block is optional. If present, the descriptor block must come after the data records, and record 2 must contain the record number of the first item in the block. Information in the descriptor block consists of *tags*, which identify the type of information that follows; followed by the number of items associated with the tag; followed by the items themselves.

TAG,number of items,item one,item two...

The information in the descriptor block can be written serially, or, if the logical record size is sufficiently large, written one tag to a record. In either case, the descriptor block must be able to be read serially.

For example, to describe the names of the columns and the fact that the units are in degrees Kelvin, the descriptor block for the above file might look like this:

Record	File Contents	Comments
67	"COLNAMS",3,"TEMP","VISCOSITY","DENSITY",	Column names: Temp, viscosity, density.
	"XYOFFSET",2,4, − 3,	X-Y offset: (4, − 3).
	"DEGREES",1,"KELVIN"	Information on units: Degrees Kelvin.
EOF		

The Curve Fitting Files

The HP-71 Curve Fitting Pac can read any HPAF file. String-type data items will be ignored automatically. For instance, if a file has a string *NSSNSNN*, columns 1, 4, 6, and 7 will be considered the first, second, third, and fourth columns in the data array for the CFIT program. The first and second columns would be interpreted as the independent variables, the third column as the dependent variable, and the fourth column as the weight.

When writing to a data file, CFIT generates no descriptor block, so the record number in record 2 is set to zero.

Appendix G

Creating Your Own Model or Function Subprogram

This appendix contains information on the use of subprograms that you can write for both function representation in OPTIMIZE and model representation in CFIT.

Writing a Model or Function Subprogram

The purpose of these subprograms is to compute the value of a function and its gradient given certain inputs. The inputs vary depending on which program calls your subprogram. If OPTIMIZE calls it, the input is an array containing the values of the function's variables. If CFIT calls it, the input is two arrays, one containing a set of parameters and the other containing data. Because the inputs differ between the two programs, there are differences in the subprogram each requires. However, in both cases the output is a function value and gradient vector.

The steps you follow to write a subprogram are the same whether you are writing it for CFIT or OPTIMIZE. The steps are as follows:

- Write the equation describing your function in the form of *dependent variable* = *function of independent variables and parameters.*
- If possible, write the partial derivatives with respect to the unknowns. (The unknowns are the variables in OPTIMIZE and the parameters in CFIT.)
- Optionally modify the equations in order to make execution faster.
- Associate the symbols in your equation with the variables in the standard subprogram syntax for your application (either CFIT or OPTIMIZE).
- Write the subprogram.
- Design and add any special error checking or other features you want.
- Check your work and verify your subprogram by using it on a sample problem with known input and output values for comparison.

Standard Subprogram Syntax

Here is the standard syntax for user subprograms called by any of the programs in this pac.

For OPTIMIZE:

SUB *name*(P(),G(),F,C)

Variables	Use
Inputs:	
P()	The current value of the variables.
Outputs:	
G()	The gradient of the function at the current variables.
F	The function value.
C	The condition code.

For CFIT:

SUB *name*(P(),X(),G(),F,C)

Variables	Use
Inputs:	
P()	The current value of the model parameters.
X()	The independent variables. (The dimension must also allow for the dependent variable and the weight.)
Outputs:	
G()	The model gradient with respect to the unknown parameters.
F	The model value.
C	The condition code.

Speed and Accuracy

Gradient

Including the gradient calculation in your subprogram is optional—the main programs can approximate the gradient for you. However, if you can include the gradient calculation in your subprogram, you can improve performance of your subprogram in terms of both speed and accuracy. For an explanation of how the FP Method uses the gradient calculation, refer to "A Word on Gradient," page 61.

> **Note:** If you write your own subprogram for a trigonometric function and *include the gradient calculation*, you need to make sure that your subprogram agrees with the machine setting of degrees or radians. It's usually best to declare in your subprogram which mode you want. Also, when you get your results, be sure to interpret them according to the mode the machine is in.

Speed Tricks

Anything you can do to increase the execution speed of your subprogram is helpful (especially on long, time consuming problems) due to the number of times CFIT and OPTIMIZE call a subprogram in the iteration process. A list of a few time-saving tricks follows, and, with experience, you may develop your own techniques to extend the list given here.

- Substitute multiplication for exponentiation when possible (for example, use A*A rather than A^2).
- Examine the equations for F and $G(\ )$ to see if computing all or part of them in a different order will reduce the total number of math operations.
- If a group of operations occurs repeatedly in your subprogram, calculate it once and assign it to some intermediate variable, then use that variable in place of the operations.

Important Interface Assumptions

Condition Code

The condition code variable (C) allows a user-written subprogram to indicate that something has gone wrong. The programs in this pac always set C equal to zero before calling a user-written subprogram and always test it on return. If the condition code is found to be zero, it indicates there were no exceptional events; if it is found to be positive, it indicates an error; and if it is found to be negative, it indicates a warning. Refer to appendix B, starting on page 69, for more information on the relationship between condition codes and error messages.

Option Base and Option Round

Both CFIT and OPTIMIZE require you to write your subprogram assuming the base option for dimensioning arrays is set to 1 and the rounding option is set to OPTION ROUND NEAR (the default rounding mode).

What You Should Not Do

What follows is a short list of items that you should avoid doing in your subprograms.

- *You should not change the values of inputs to your subprogram unless you restore the original values on exit.*

  The gradient approximation routines assume that the point at which the gradient is being approximated does not change through several calls to your subprogram. If a change to the *P*( ) or *X*( ) variables is made within your subprogram, these routines will not return correct results.

- *You should not change certain flag values unless you restore the original values on exit.*

  Flags 61, 62, and 63 are inputs to some of the computational routines, specifically GRADF, GRADM, CSQ, FIT, and FP. Also, flags 57-60 are used by OPTIMIZE and CFIT. (Although flags 57-63 are used by the curve fitting programs, their original values are restored when the programs end.) If changed within your subprogram and not restored prior to exit, they may give you erroneous results.

- *You should not make certain recursive calls.*

  Recursive calls are calls to subprograms with passed parameters that will cause it, or something it calls, to call your subprogram.

  Some of the binary subprograms in the file FITLIB are *not* recursive since they use a dedicated buffer for intermediate computations. These are GRADF, GRADM, CSQ, FP, and FIT. For example, it is acceptable for your subprogram to call POLY to evaluate a polynomial, but your subprogram (MYSUB) in your file (MYFILE) should not execute the call

  ```
  CALL GRADF("MYSUB","MYFILE",P(),G(),F,Z)
  ```

  This recursive situation has no useful value and will eventually result in an insufficient memory error as the two programs keep calling each other indefinitely.

Example Subprogram for OPTIMIZE

The steps to go through in the process of developing a subprogram for OPTIMIZE are outlined next in the form of an example.

Function

$$F1(a,b) = a^2 - 2ab + 2b^2$$

Gradient

$$\nabla F1(a,b) = \begin{bmatrix} 2(a - b) \\ 4b - 2a \end{bmatrix}$$

Subprogram

```
10 Sub F1(P(),G(),F,C)
20 F=P(1)*P(1)-2*P(1)*P(2)+2*P(2)*P(2)  !  Function Value
30 G(1)=2*(P(1)-P(2))                   !  2(a-b)
40 G(2)=2*P(2)-G(1)                     !  2b-(2(a-b))=4b-2a
50 ENDSUB
```

Variable Usage

Variable	Use
P()	The current value of the variables (a and b).
G()	The gradient of $F1$ at the current variables.
F	The function value.
C	The condition code.

Comments

A few comments about the subprogram are listed below:

- The computation in line 40 saves one multiplication over the more straight-forward alternative (G(2)=4*P(2)-2*P(1)).
- This example does not include error handling within the subprogram. However, even this simple subprogram can encounter exceptional conditions with certain arguments (for example, overflow and underflow).
- The function $F1$ achieves its minimum value of 0 at $a = b = 0$. (You might want to try this example with OPTIMIZE.)

Example Subprogram for CFIT

The steps to go through in the process of developing a subprogram for CFIT are outlined next in the form of an example.

Model

$P(V) = C \times V^{-N}$

$PV^N = C$ describes the pressure-volume relationship of a certain system during heating. The model equation was derived from this equation by rewriting it to get the dependent variable alone on the left. The independent variable is V, the dependent variable is P, and the values of N and C are model parameters to be determined based on data collected for P as a function of V.

Gradient

$$\nabla P(C,N) = \begin{bmatrix} \partial P/\partial C \\ \partial P/\partial N \end{bmatrix} = \begin{bmatrix} V^{(-N)} \\ -\ \mathrm{LN}(V) \times P(V) \end{bmatrix}$$

Subprogram

```
10  Sub PRESSURE(P(),X(),G(),F,C)
20  G(1)=X(1)^(-P(2))                  !  V^(-N)
30  F=P(1)*G(1)                        !  P=C*V^(-N)
40  G(2)=-LN(X(1))*F                   !  -LN(V)*P(V)
50  ENDSUB
```

Variable Usage

Variable	Use
P()	The current value of the model parameters (C and N).
X()	The independent variables (V).
G()	The model gradient with respect to the unknown parameters.
F	The model value.
C	The condition code.

Comments

A few comments about the subprogram are listed below:

- The order of computation within the subprogram requires only one exponential evaluation, thus providing for faster execution.
- This example does not include error handling within the subprogram (with reasonable data, exceptional conditions are not likely).
- One approach to minimize the pain of translating your variables into those used by your subprogram follows:

  1. Declare variables you want to use within your subprogram.
  2. Assign to your own variables those input values passed in through the parameter list.
  3. Perform the necessary computations using your variables.
  4. Store the results into the output parameters as required.

  For a good example of this approach, refer to the "Big Box" example on page 22. In this example, the computations were done with the declared variables *W* and *H*.

- Because it differs from the previous subprogram only in the sign of $P(2)$, you could have used the subprogram `POWER` in the library file `MODELS` for the model. All you would need to do to use it is negate the value returned for $P(2)$ when you get the results. Don't forget the built-in models!
- Actually, with two reliable observations, (V_1, P_1) and (V_2, P_2), this problem can be solved directly and has the solution shown here:

$$N = \frac{\ln(V_2/V_1)}{\ln(P_1/P_2)}, \qquad C = P_1 \times V_1^{\,N}$$

  Often, however, a good fit over a range of observations will result in a curve that more accurately reflects your entire data set than that obtained by a direct solution using limited data.

Appendix H

File Names Used in This Pac

This appendix contains a list of the file names used in this pac. All of these files except one can be copied to main memory, assuming you have enough memory available. (CurveFit is the only file that cannot be copied.) Most of these files are in ROM and, since your HP-71 searches main RAM first when it looks for a file name, be sure to *not* have any files with the following names in main RAM when you are using the curve fitting module.

File Name	**File Type**	**Description**
CurveFit	LEX	Contains the pac identifier.
CFIT	BASIC	Contains the program for curve fitting.
OPTIMIZE	BASIC	Contains the program for optimizing.
MODELS	BASIC	Contains the BASIC curve fitting models supplied by the pac.
PCENTCHI	BASIC	Evaluates the left tail of the Chi Square density function.
FITLEX	LEX	Contains the keywords KEYWAIT$ and KILLBUFF, the message table used by the binary subprograms, and a routine that saves scratch memory used by the pac when your machine is turned off.
FITLIB	BIN	Contains all binary user-accessible subprograms.
CFKEYZ	KEY	Contains key definitions for the CFIT array editor.
USERKEYS*	KEY	Saves previously user-defined keys while the pac is running.
CFKEYS*	KEY	Contains a copy in RAM of CFKEYZ.

* A reserved file name created in RAM when CFIT is running. If you already have a file with this name in RAM, it will be destroyed when you run CFIT.

Appendix I

Glossary

A,B

bound estimate: A control input to CFIT and OPTIMIZE that estimates a local minimum or maximum value for the function to be optimized. This value is used in the line search algorithm, and, if chosen appropriately, can improve performance.

C

Chi Square (χ^2): The function whose value is the sum, over all data, of the squares of the weighted differences $(Y - F)/W$ of the dependent variables Y and the model F. If the Y's are normally distributed with mean F and variance W^2, then Chi Square is $\chi^2(\nu)$ distributed with ν (the number of data points minus the number of model parameters) degrees of freedom.

condition code: A value assigned to the condition code variable.

condition code variable: A variable passed by reference to a subprogram for the purpose of indicating to the calling program the nature of any exceptional event encountered during execution of the subprogram.

controls: The set of inputs used by the computational routines called by CFIT and OPTIMIZE.

converge: The condition in which the graph of the function being optimized is "sufficiently flat" as measured by the gradient norm to stop the iteration process.

D,E

Delta: A control input to CFIT, used in gradient approximation. It is described in CFIT as a "constant" or "percentage."

dependent variable: The measured or Y value that depends on the independent variables (X's) in the data set.

F

Fletcher-Powell Method: An optimization algorithm introduced by R. Fletcher and M. J. D. Powell.

G

gradient: Given a function $F(p_1,p_2,\ldots,p_k)$, the gradient of F is a vector-valued function whose value at $(p_1,p_2,\ldots,p_k)$ is the vector having the partial derivatives of F with respect to the variables $(p_1,p_2,\ldots,p_k)$ as coordinates.

gradient norm: With a gradient vector $(g_1,g_2,\ldots,g_k)$, the gradient norm is $\sqrt{(g_1^2 + g_2^2 + \ldots + g_k^2)}$. This value appears in CFIT and OPTIMIZE output identified as |Grd|.

H

HPAF format: Hewlett-Packard Application File format. It refers to a standard DATA file format used by HP-71 application pacs.

I,J,K

independent variable: A variable controlled by the experimenter. It usually represents a variable whose value is selected rather than measured.

iterate: A value produced by an iterative method.

iteration: The step number in an iterative method (for example, the 12th iteration).

iterative method: A method that determines a succession of values where each successive value is dependent on one or more of the previous values.

L

line search: The algorithmic attempt to minimize a function of one or more variables restricted to a particular direction (for example, the attempt to minimize $Z = X^2 + Y^2 + (Y - 3)^2$ along the line $Y = X + 1$).

M,N

model function: A function $F = F(x_1,x_2,\ldots,x_n;\ p_1,p_2,\ldots,p_k)$ of $n + k$ variables. The p's are parameters to be determined to best fit the data. The x's are the independent variables in the data. Model functions are represented in the pac by subprograms.

O

optimize: The attempt to locate critical points of a function—in particular, local maxima and minima.

P,Q,R

parameter: One of the unknown values that determine the model. The task of CFIT is to produce the "best" values for the model parameters.

percentage goodness of fit: A model evaluation aid equal to 100 × $(1 - P)$ where P is the value returned by the subprogram PCENTCHI. Under appropriate conditions this value may be used as a model rejection criterion.

S,T,U

scalar: A numeric value. It is used to distinguish numeric values from vectors.

scratch memory: The memory used by the numeric computation routines (binary subprograms) to store arguments and intermediate results during the computations.

V

vector: A vector of length k is an ordered list $(p_1, p_2, \ldots, p_k)$. In this pac, the list elements are scalar values.

W,X,Y,Z

weight: A value associated with a data point to provide greater or lesser significance to the point in the curve fit process. Relatively large weights correspond to low significance. In order for percentage goodness of fit to have statistical significance, the weights should equal the standard deviations of the dependent variables.

Index

Page numbers in **bold** indicate primary reference; page numbers in regular type indicate secondary references.

HEWLETT
PACKARD

Portable Computer Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

European Headquarters
150, Route du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva - Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL

82484-90001 English

Printed in Singapore 3/84