

## NAG Library Chapter Introduction

### P01 – Error Trapping

#### Contents

<b>1</b>	<b>Scope of the Chapter</b> .....	2
<b>2</b>	<b>Background to the Problems</b> .....	2
2.1	Errors, Failure and Warning Conditions .....	2
2.2	The IFAIL Parameter .....	2
2.3	Hard Fail Option .....	2
2.4	Soft Fail Option .....	3
2.5	Historical Note .....	3
<b>3</b>	<b>Recommendations on Choice and Use of Available Routines</b> .....	4
<b>4</b>	<b>Routines Withdrawn or Scheduled for Withdrawal</b> .....	4

## 1 Scope of the Chapter

This chapter is concerned with the trapping of error, failure or warning conditions by NAG Library routines. This introduction document describes the commonly occurring parameter IFAIL.

## 2 Background to the Problems

### 2.1 Errors, Failure and Warning Conditions

The error, failure or warning conditions considered here are those that can be detected by explicit coding in a Library routine. Such conditions must be anticipated by the author of the routine. They should not be confused with run-time errors detected by the compiling system, e.g., detection of overflow or failure to assign an initial value to a variable.

In the rest of this document we use the word ‘error’ to cover all types of error, failure or warning conditions detected by the routine. They fall roughly into three classes.

- (i) On entry to the routine the value of a parameter is out of range. This means that it is not useful, or perhaps even meaningful, to begin computation.
- (ii) During computation the routine decides that it cannot yield the desired results, and indicates a failure condition. For example, a matrix inversion routine will indicate a failure condition if it considers that the matrix is singular and so cannot be inverted.
- (iii) Although the routine completes the computation and returns results, it cannot guarantee that the results are completely reliable; it therefore returns a warning. For example, an optimization routine may return a warning if it cannot guarantee that it has found a local minimum.

**All three classes of errors are handled in the same way by the Library.**

Each error which can be detected by a Library routine is associated with a number. These numbers, with explanations of the errors, are listed in Section 6 (Error Indicators and Warnings) in the routine document. Unless the document specifically states to the contrary, you should not assume that the routine necessarily tests for the occurrence of the errors in their order of error number, i.e., the detection of an error does not imply that other errors have or have not been detected.

### 2.2 The IFAIL Parameter

Most of the NAG Library routines which can be called directly by you have a parameter called IFAIL. This parameter is concerned with the NAG Library error trapping mechanism (and, for some routines, with controlling the output of error messages and advisory messages).

IFAIL has **two** purposes:

- (i) to allow you to specify what action the Library routine should take if an error is detected;
- (ii) to inform you of the outcome of the call of the routine.

For purpose (i), you **must** assign a value to IFAIL before the call to the Library routine. Since IFAIL is reset by the routine for purpose (ii), the parameter must be the name of a variable, **not** a literal or constant.

The value assigned to IFAIL before entry should be either 0 (**hard fail** option), or 1 or – 1 (**soft fail** option). If after completing its computation the routine has not detected an error, IFAIL is reset to 0 to indicate a **successful call**. Control returns to the calling program in the normal way. If the routine does detect an error, its action depends on whether the hard or soft fail option was chosen.

### 2.3 Hard Fail Option

If you set IFAIL to 0 before calling the Library routine, execution of the program will terminate if the routine detects an error. Before the program is stopped, this error message is output:

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: IFAIL = n
** NAG hard failure - execution terminated
```

where XXXXXX is the routine name, and n is the number associated with the detected error. An explanation of error number n is given in Section 6 of the routine document XXXXXX.

In addition, most routines output explanatory error messages immediately before the standard termination message shown above.

The hard fail option should be selected if you are in any doubt about continuing the execution of the program after an unsuccessful call to a NAG Library routine. For environments where it might be inappropriate to halt program execution when an error is detected it is recommended that the hard fail option is **not** used.

## 2.4 Soft Fail Option

To select this option, you must set IFAIL to 1 or –1 before calling the Library routine.

If the routine detects an error, IFAIL is reset to the associated error number; further computation within the routine is suspended and control returns to the calling program.

If you set IFAIL to 1, then no error message is output (**silent exit**). If the output of error messages is undesirable, then silent exit is recommended.

If you set IFAIL to –1 (**noisy exit**), then before control is returned to the calling program, the following error message is output:

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: IFAIL = n
** NAG soft failure - control returned
```

In addition, most routines output explanatory error messages immediately before the above standard message.

**It is most important to test the value of IFAIL on exit if the soft fail option is selected.** A nonzero exit value of IFAIL implies that the call was not successful so it is imperative that your program be coded to take appropriate action. That action may simply be to print IFAIL with an explanatory caption and then terminate the program. Many of the example programs in Section 9 of the routine documents have IFAIL-exit tests of this form. In the more ambitious case, where you wish your program to continue, it is essential that the program can branch to a point at which it is **sensible** to resume computation.

The soft fail option puts the onus on you to handle any errors detected by the Library routine. With the proviso that you are able to implement it **properly**, it is clearly more flexible than the hard fail option since it allows computation to continue in the case of errors. In particular there are at least two cases where its flexibility is useful:

- (i) where additional information about the error or the progress of computation is returned via some of the other parameters;
- (ii) in some routines, ‘partial’ success can be achieved, e.g., a probable solution found but not all conditions fully satisfied, so the routine returns a warning. On the basis of the advice in Section 6 and elsewhere in the routine document, you may decide that this partially successful call is adequate for certain purposes.

## 2.5 Historical Note

The error handling mechanism described above was introduced into the NAG Library at Mark 12. It supersedes the earlier mechanism which for most routines allowed IFAIL to be set by you to 0 or 1 only. The new mechanism is compatible with the old except that the details of the messages output on hard failure have changed. The new mechanism also allows you to set IFAIL to –1 (soft failure, noisy exit).

A few routines (introduced mainly at Marks 7 and 8) use IFAIL in a different way to control the output of error messages, and also of advisory messages (see Chapter X04). In those routines IFAIL is regarded as a decimal integer whose least significant digits are denoted *ba* with the following significance:

$a = 0$ : hard failure	$a = 1$ : soft failure
$b = 0$ : silent exit	$b = 1$ : noisy exit

Details are given in the documents of the relevant routines; for those routines this alternative use of IFAIL remains valid.

### 3 Recommendations on Choice and Use of Available Routines

Please note that this chapter is scheduled for withdrawal at Mark 24.

### 4 Routines Withdrawn or Scheduled for Withdrawal

<b>Withdrawn Routine</b>	<b>Mark of Withdrawal</b>	<b>Replacement Routine(s)</b>
P01ABF	24	No longer required

---