Module 1.2: nag_error_handling
Error Handling

`nag_error_handling` provides facilities for handling errors detected by NAG *fl*90 procedures.

Contents

Introduction

1 Classification of Errors

NAG *fl*90 procedures may detect and report various kinds of error, failure or warning conditions. They are classified into three levels of increasing severity.

Level 1 (Warning): a warning that, although the computation has been completed, the results may not be completely satisfactory.

Level 2 (Failure): a numerical failure during computation (for example, failure of an iterative algorithm to converge).

Level 3 (Fatal): a fatal error which prevents the procedure from attempting any computation (for example, invalid input arguments, or failure to allocate enough memory).

Each error is given a numeric code. Warnings (level 1) have codes in the range 100–199; failures (level 2) have codes in the range 200–299; fatal errors (level 3) have codes in the range 300–399.

To control the way in which any of these errors are handled, all NAG *fl*90 procedures have an *optional* argument `error` (except for a few which cannot give rise to any error condition).

2 Default Error Handling

If the optional argument `error` is *omitted*, then the Library procedure takes the following *default* action.

- If no error is detected, it returns control to your calling program.
- If it detects an error of level 1 (warning), it writes an error-message to the standard output unit and returns control to your calling program.
- If it detects an error of level 2 or 3 (failure in computation or fatal error), it writes an error message to the standard output unit and halts execution of the program.

3 Non-default Error Handling

The default action may not always be suitable: for example, you may wish to take corrective action after a computational failure, or you may wish to halt execution after a warning. In such cases, you must supply the optional argument `error`. It is a structure of a derived type `nag_error`, defined by the Library. It is an argument of intent(inout), which serves two functions.

1. It allows you to specify what action a Library procedure should take if it detects an error.
2. It reports the state of the Library procedure (either success or an error-condition) to your calling program, and returns the text of any error message.

If you supply the argument `error`:

- You *must initialize* it by calling the procedure `nag_set_error`.
- You *must test* one of the integer components `code` or `level` of the structure `error` on return to your calling program; if you do not, your program may continue computing with invalid or undefined results.

You do not normally need to include any additional `USE` statements in your calling program in order to use the derived type `nag_error` and the procedure `nag_set_error`; they are always accessible through any `USE` statement which gives access to a Library procedure with the optional argument `error`. They can also be accessed by the statement:

```
USE nag_error_handling
```

Procedure: nag_set_error

1 Description

nag_set_error assigns values to some of the components of a structure of type nag_error, in order to control the way in which errors are handled by a NAG *fl*90 procedure.

Each call to this procedure reinitializes the structure; components which have been set by a previous call to this procedure are not preserved in subsequent calls.

2 Usage

```
USE nag_error_handling
CALL nag_set_error(error  [, optional arguments])
```

3 Arguments

3.1 Mandatory Argument

error — type(nag_error), intent(out)

Output: the error handling structure, initialized so that it can be passed to a Library procedure.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

halt_level — integer, intent(in), optional

Input: the value to be assigned to the component error%halt_level. It specifies that the program is to be halted if an error of level $\geq$ halt_level is detected.

Note: be very careful if you use a value ≥ 4 for halt_level: Library procedures will then return control to the calling program even after a fatal error, and the program *must* test for *all* types of error and take suitable action.

Default: halt_level = 2.

print_level — integer, intent(in), optional

Input: the value to be assigned to the component error%print_level. It specifies that an error message is to be printed if an error of level $\geq$ print_level is detected. If print_level = 0, a message is printed to report successful exit from a Library procedure.

Note: it is usually sensible to ensure that print_level $\leq$ halt_level, so that an error message is always printed when a Library procedure halts the program.

Default: print_level = 1.

unit — integer, intent(in), optional

Input: the value to be assigned to the component error%unit. It specifies the unit on which any error message is to be printed.

Default: unit = nag_std_out (the default output unit in your implementation of the Library).

4 Error Codes

None.

5 Examples of Usage

Complete examples of the use of this procedure appear in Examples 1 and 2 of this module document.

Derived Type: nag_error

1 Description

The derived type `nag_error` is used to communicate information about error handling between a user's program and a NAG *fl*90 procedure.

2 Type Definition

```
type nag_error
  integer::  halt_level
  integer::  print_level
  integer::  unit
  integer::  code
  integer::  level
  character(len=31) ::  generic_name
  integer::  msg_len
  character(len=79) ::  msg(12)
end type nag_error
```

3 Components

The public components of the type are listed below.

halt_level — integer
: Specifies that the program is to be halted if an error of level $\geq$ `halt_level` is detected.

print_level — integer
: Specifies that an error message is to be printed if an error of level $\geq$ `print_level` is detected.

unit — integer
: The unit number of the file to which any error message is to be written.

code — integer
: The error code if an error has been detected, or 0 if the procedure has exited successfully.

level — integer
: The error level if an error has been detected, or 0 if the procedure has exited successfully.

generic_name — character(len=31)
: The generic procedure name.

msg_len — integer
: The number of elements of the component `msg` which are used to store the text of the error message. $1 \leq$ `msg_len` ≤ 12.

msg(12) — character(len=79)
: The text of the error message in elements 1: `msg_len`.

Example 1: Illustration of Each Level of Error Exit

This example illustrates all three levels of error exit from a Library procedure. The procedure is `nag_gen_lin_sol`, which solves a system of linear equations. For the purposes of this example, a trivial system with a diagonal coefficient matrix is solved.

In the program below, `nag_gen_lin_sol` is called four times. The first call is successful, and no error exit occurs. For the next two calls the data is modified, so that first a warning is raised, and then a failure is reported. For the last call, an invalid value of an argument is supplied, forcing a fatal error. The results show the error messages produced by the Library procedure after each error exit.

Before each call to `nag_gen_lin_sol`, there is a call to `nag_set_error` which specifies that the Library procedure will halt the program only after a fatal error. In the final call to `nag_gen_lin_sol` a fatal error does occur and the program is halted.

Since the same structure `error` is passed to `nag_gen_lin_sol` in each call, it would be sufficient to call `nag_set_error` once at the beginning of the program. However, it is safe practice, if you use non-default error handling, to place the call to `nag_set_error` immediately before the call to the Library procedure.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_error_handling_ex01

  ! Example Program Text for nag_error_handling
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_gen_lin_sys, ONLY : nag_gen_lin_sol, nag_error, nag_set_error
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC EPSILON, KIND, REAL
  ! .. Parameters ..
  INTEGER, PARAMETER :: n = 3
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i
  TYPE (nag_error) :: error
  ! .. Local Arrays ..
  REAL (wp) :: a(n,n), aa(n,n), b(n), bb(n)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) &
   'Example Program Results for nag_error_handling_ex01'
  WRITE (nag_std_out,*)

  ! Initialize the arrays a and b for passing to nag_gen_lin_sol.

  a = 0.0_wp
  DO i = 1, n
    a(i,i) = REAL(i,kind=wp)
  END DO
  b = 1.0_wp

  WRITE (nag_std_out,*) '1st call to nag_gen_lin_sol'

  ! Straightfoward problem: successful exit

  aa = a
```

```
bb = b

CALL nag_set_error(error,halt_level=3)

CALL nag_gen_lin_sol(a,b,error=error)

WRITE (nag_std_out,'(1x,2(a,i3))') &
 'On exit from nag_gen_lin_sol, error%code =', error%code, &
 '  error%level =', error%level

IF (error%level<=1) THEN
  WRITE (nag_std_out,*) 'Solution returned in b:'
  WRITE (nag_std_out,'(1X,6E11.3)') b
ELSE
  WRITE (nag_std_out,*) 'No solution returned in b'
END IF

WRITE (nag_std_out,*)
WRITE (nag_std_out,*)
WRITE (nag_std_out,*) '2nd call to nag_gen_lin_sol'

! Very ill-conditioned problem: a warning is raised

a = aa
b = bb
! modify A to force a warning
a(1,1) = 0.1_wp*EPSILON(1.0_wp)

CALL nag_set_error(error,halt_level=3)

CALL nag_gen_lin_sol(a,b,error=error)

WRITE (nag_std_out,'(1x,2(a,i3))') &
 'On exit from nag_gen_lin_sol, error%code =', error%code, &
 '  error%level =', error%level

IF (error%level<=1) THEN
  WRITE (nag_std_out,*) 'Solution returned in b:'
  WRITE (nag_std_out,'(1X,6E11.3)') b
ELSE
  WRITE (nag_std_out,*) 'No solution returned in b'
END IF

WRITE (nag_std_out,*)
WRITE (nag_std_out,*)
WRITE (nag_std_out,*) '3rd call to nag_gen_lin_sol'

! Singular problem: a failure is reported

a = aa
b = bb
! modify A to force a failure
a(1,1) = 0.0_wp

CALL nag_set_error(error,halt_level=3)

CALL nag_gen_lin_sol(a,b,error=error)

WRITE (nag_std_out,'(1x,2(a,i3))') &
 'On exit from nag_gen_lin_sol, error%code =', error%code, &
 '  error%level =', error%level
```

```
      IF (error%level<=1) THEN
        WRITE (nag_std_out,*) 'Solution returned in b:'
        WRITE (nag_std_out,'(1X,6E11.3)') b
      ELSE
        WRITE (nag_std_out,*) 'No solution returned in b'
      END IF

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) '4th call to nag_gen_lin_sol'

      ! Invalid value supplied for trans: a fatal error occurs

      a = aa
      b = bb

      CALL nag_set_error(error,halt_level=3)

      CALL nag_gen_lin_sol(a,b,trans='s',error=error)

      WRITE (nag_std_out,'(1x,2(a,i3))') &
       'On exit from nag_gen_lin_sol, error%code =', error%code, &
       '  error%level =', error%level
      IF (error%level<=1) THEN
        WRITE (nag_std_out,*) 'Solution returned in b:'
        WRITE (nag_std_out,'(1X,6E11.3)') b
      ELSE
        WRITE (nag_std_out,*) 'No solution returned in b'
      END IF

    END PROGRAM nag_error_handling_ex01
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_error_handling_ex01

 1st call to nag_gen_lin_sol
 On exit from nag_gen_lin_sol, error%code =  0  error%level =  0
 Solution returned in b:
   0.100E+01  0.500E+00  0.333E+00


 2nd call to nag_gen_lin_sol
 ****************** Warning reported by NAG Fortran 90 Library *****************
 Procedure nag_gen_lin_sol                  Level = 1  Code = 101
 The matrix of the coefficients is nearly singular.
 Reciprocal of the condition number =  7.401486830834377E-018 .
 The solution may have no accuracy at all.
 Examine the estimate of the forward error that may be returned in fwd_err.
 **************************** Execution continued *****************************
 On exit from nag_gen_lin_sol, error%code =101  error%level =  1
 Solution returned in b:
   0.450E+17  0.500E+00  0.333E+00


 3rd call to nag_gen_lin_sol
 ****************** Failure reported by NAG Fortran 90 Library *****************
```

```
Procedure nag_gen_lin_sol                  Level = 2  Code = 201
The matrix of the coefficients A is exactly singular.
The factor U is exactly singular: U(i,i) is zero for i =           1;
the factorization has been completed but if it is used to solve a system of
equations, an error will occur.
*************************** Execution continued ******************************
On exit from nag_gen_lin_sol, error%code =201  error%level =  2
No solution returned in b


4th call to nag_gen_lin_sol
**************** Fatal error reported by NAG Fortran 90 Library ***************
Procedure nag_gen_lin_sol                  Level = 3  Code = 301
An input argument has an invalid value.
trans = 's'
trans must be 'N', 'n', 'T', 't', 'C' or 'c'.
**************************** Execution halted *******************************
```

Example 2: Illustration of Different Ways to Handle Warnings

This example illustrates three different ways of handling warnings raised by a Library procedure. The procedure is `nag_gen_lin_sol`, which solves a system of linear equations and raises a warning if the system is very ill conditioned.

In the program below, `nag_gen_lin_sol` is called three times, each time to solve the same very ill conditioned system. The code is the same in all three cases, except for the calls to `nag_set_error`; these are explained by comments in the code.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_error_handling_ex02

  ! Example Program Text for nag_error_handling
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_out
  USE nag_gen_lin_sys, ONLY : nag_gen_lin_sol, nag_error, nag_set_error
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC EPSILON, KIND, REAL
  ! .. Parameters ..
  INTEGER, PARAMETER :: n = 3
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i
  TYPE (nag_error) :: error
  ! .. Local Arrays ..
  REAL (wp) :: a(n,n), aa(n,n), b(n), bb(n)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) &
   'Example Program Results for nag_error_handling_ex02'
  WRITE (nag_std_out,*)

  ! Initialize A to a diagonal matrix and b to (1, 1, ...,  1)

  a = 0.0_wp
  DO i = 1, n
    a(i,i) = REAL(i,kind=wp)
  END DO
  a(1,1) = 0.1_wp*EPSILON(1.0_wp)
  b = 1.0_wp

  WRITE (nag_std_out,*) '1st call to nag_gen_lin_sol'

  aa = a
  bb = b

  ! The optional argument error is passed to nag_gen_lin_sol, so
  ! that the program can test if a warning is raised and take
  ! special action.
  ! nag_set_error must be called to initialize error, but default
  ! values are used for halt_level and print_level
```

```
      CALL nag_set_error(error)

      CALL nag_gen_lin_sol(a,b,error=error)

      WRITE (nag_std_out,'(1x,2(a,i3))') &
       'On exit from nag_gen_lin_sol, error%code =', error%code, &
       '  error%level =', error%level

      IF (error%level==1) THEN
        WRITE (nag_std_out,*) &
         'Special action can be taken here if a warning occurs'
      END IF

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) '2nd call to nag_gen_lin_sol'

      a = aa
      b = bb

      ! The next call is the same as the first, except that we suppress
      ! the printing of warning messages by specifying a print_level of
      ! 2.

      CALL nag_set_error(error,print_level=2)

      CALL nag_gen_lin_sol(a,b,error=error)

      WRITE (nag_std_out,'(1x,2(a,i3))') &
       'On exit from nag_gen_lin_sol, error%code =', error%code, &
       '  error%level =', error%level

      IF (error%level==1) THEN
        WRITE (nag_std_out,*) &
         'Special action can be taken here if a warning occurs'
      END IF

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) '3rd call to nag_gen_lin_sol'

      a = aa
      b = bb

      ! For the next call, we specify a halt_level of 1, so that the
      ! program will halt if a warning is raised. Since a warning does
      ! occur, the statements after the call to nag_gen_lin_sol are not
      ! executed.

      CALL nag_set_error(error,halt_level=1)

      CALL nag_gen_lin_sol(a,b,error=error)

      WRITE (nag_std_out,'(1x,2(a,i3))') &
       'On exit from nag_gen_lin_sol, error%code =', error%code, &
       '  error%level =', error%level

      IF (error%level==1) THEN
        WRITE (nag_std_out,*) &
         'Special action can be taken here if a warning occurs'
      END IF
```

```
   END PROGRAM nag_error_handling_ex02
```

2 Program Data

None.

3 Program Results

```
 Example Program Results for nag_error_handling_ex02

 1st call to nag_gen_lin_sol
 ****************** Warning reported by NAG Fortran 90 Library *****************
 Procedure nag_gen_lin_sol                  Level = 1  Code = 101
 The matrix of the coefficients is nearly singular.
 Reciprocal of the condition number =  7.401486830834377E-018 .
 The solution may have no accuracy at all.
 Examine the estimate of the forward error that may be returned in fwd_err.
 **************************** Execution continued ******************************
 On exit from nag_gen_lin_sol, error%code =101  error%level =  1
 Special action can be taken here if a warning occurs


 2nd call to nag_gen_lin_sol
 On exit from nag_gen_lin_sol, error%code =101  error%level =  1
 Special action can be taken here if a warning occurs


 3rd call to nag_gen_lin_sol
 ****************** Warning reported by NAG Fortran 90 Library *****************
 Procedure nag_gen_lin_sol                  Level = 1  Code = 101
 The matrix of the coefficients is nearly singular.
 Reciprocal of the condition number =  7.401486830834377E-018 .
 The solution may have no accuracy at all.
 Examine the estimate of the forward error that may be returned in fwd_err.
 ***************************** Execution halted ********************************
```