GALAHAD FDH

USER DOCUMENTATION GALAHAD Optimization Library version 5.0

1 SUMMARY

This package **computes a finite-difference approximation to the Hessian matrix $\mathbf{H}(\mathbf{x})$**, for which $(\mathbf{H}(\mathbf{x}))_{i,j} = \partial f^2/\partial x_i \partial x_j$, $1 \leq i,j \leq n$, using values of the gradient $\mathbf{g}(\mathbf{x}) = \nabla_x f(\mathbf{x})$ of the function $f(\mathbf{x})$ of n unknowns $\mathbf{x} = (x_1,\ldots,x_n)^T$. The method takes advantage of the entries in the Hessian that are known to be zero. The user must specify the step sizes to be used in the finite difference calculation and either supply a routine to evaluate the gradient or provide gradient values by reverse communication.

ATTRIBUTES — Versions: `GALAHAD_FDH_single`, `GALAHAD_FDH_double`. **Uses:** `GALAHAD_SYMBOLS`, `GALAHAD_SPECFILE`, `GALAHAD_SPACE` and `GALAHAD_NLPT`. **Date:** July 2012. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory, and Ph. L. Toint, The University of Namur, Belgium. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

2 HOW TO USE THE PACKAGE

The package is available using both single and double precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE GALAHAD_FDH_single
```

with the obvious substitution `GALAHAD_FDH_double`, `GALAHAD_FDH_single_64` and `GALAHAD_FDH_double_64` for the other variants.

The user is **strongly advised** to use the double precision version unless single precision corresponds to 8-byte arithmetic.

If it is required to use more than one of the modules at the same time, the derived types `FDH_control_type`, `FDH_inform_type`, `FDH_data_type` and `NLPT_userdata_type`, (Section 2.2) and the subroutines `FDH_initialize`, `FDH_analyse`, `FDH_estimate`, `FDH_terminate`, (Section 2.3) and `FDH_read_specfile` (Section 2.6) must be renamed on one of the `USE` statements.

2.1 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords `REAL(rp_)` and `INTEGER(ip_)`, where `rp_` and `ip_` are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default `REAL` for the single precision versions and `DOUBLE PRECISION` for the double precision cases, and correspond to `rp_ = real32` and `rp_ = real64`, respectively, as supplied by the fortran `iso_fortran_env` module. The latter are default (32-bit) and long (64-bit) integers, and correspond to `ip_ = int32` and `ip_ = int64`, respectively, again from the `iso_fortran_env` module.

2.2 The derived data types

Four derived data types are accessible from the package.

2.2.1 The derived data type for holding control parameters

The derived data type FDH_control_type is used to hold controlling data. Default values may be obtained by calling FDH_initialize (see Section 2.3.1), while components may also be changed by calling GALAHAD_FDH_read_spec (see Section 2.6.1). The components of FDH_control_type are:

error is a scalar variable of type INTEGER(ip_), that holds the stream number for error messages. Printing of error messages in FDH_analyse, FDH_estimate and FDH_terminate is suppressed if error ≤ 0. The default is error = 6.

out is a scalar variable of type INTEGER(ip_), that holds the stream number for informational messages. Printing of informational messages in FDH_analyse and FDH_estimate is suppressed if out < 0. The default is out = 6.

print_level is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output which is required. No informational output will occur if print_level ≤ 0. If print_level > 01, details of any data errors encountered will be reported. The default is print_level = 0.

space_critical is a scalar variable of type default LOGICAL, that must be set .TRUE. if space is critical when allocating arrays and .FALSE. otherwise. The package may run faster if space_critical is .FALSE. but at the possible expense of a larger storage requirement. The default is space_critical = .FALSE..

deallocate_error_fatal is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user wishes to terminate execution if a deallocation fails, and .FALSE. if an attempt to continue will be made. The default is deallocate_error_fatal = .FALSE..

prefix is a scalar variable of type default CHARACTER and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string prefix(2:LEN(TRIM(prefix))-1), thus ignoreing the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default prefix = "".

2.2.2 The derived data type for holding informational parameters

The derived data type FDH_inform_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of FDH_inform_type are:

status is a scalar variable of type INTEGER(ip_), that gives the exit status of the algorithm. See Sections 2.4 and 2.5 for details.

alloc_status is a scalar variable of type INTEGER(ip_), that gives the status of the last attempted array allocation or deallocation. This will be 0 if status = 0.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if status = 0.

bad_row is a scalar variable of type INTEGER(ip_), that holds the index of the first row in which inconsistent data occurred (or 0 if the data is consistent).

products is a scalar variable of type INTEGER(ip_), that gives the number of gradient evaluations (to be) used to estimate the Hessian.

2.2.3 The derived data type for holding problem data

The derived data type FDH_data_type is used to hold all the data for a particular problem, or sequences of problems with the same structure, between calls of FDH procedures. This data should be preserved, untouched (except as directed on return from GALAHAD_FDH_analyse with positive values of inform%status, see Section 2.4), from the initial call to FDH_initialize to the final call to FDH_terminate.

2.2.4 The derived data type for holding user data

The derived data type NLPT_userdata_type is available to allow the user to pass data to and from user-supplied subroutines for function and derivative calculations (see Section 2.3.5). Components of variables of type NLPT_userdata_type may be allocated as necessary. The following components are available:

integer is a rank-one allocatable array of type INTEGER(ip_).

real is a rank-one allocatable array of type default REAL(rp_)

complex is a rank-one allocatable array of type default COMPLEX (double precision complex in GALAHAD_FDH_double).

character is a rank-one allocatable array of type default CHARACTER.

logical is a rank-one allocatable array of type default LOGICAL.

integer_pointer is a rank-one pointer array of type INTEGER(ip_).

real_pointer is a rank-one pointer array of type default REAL(rp_)

complex_pointer is a rank-one pointer array of type default COMPLEX (double precision complex in GALAHAD_FDH_double).

character_pointer is a rank-one pointer array of type default CHARACTER.

logical_pointer is a rank-one pointer array of type default LOGICAL.

2.3 Argument lists and calling sequences

There are four procedures for user calls (see Section 2.6 for further features):

1. The subroutine FDH_initialize is used to set default values, and initialize private data, before solving one or more problems with the same sparsity and bound structure.
2. The subroutine FDH_analyse is called to analyze the sparsity pattern of the Hessian and to generate information that will be used when estimating its values.
3. The subroutine FDH_estimate is called repeatedly to estimate the Hessian by finite differences at one or more given points.
4. The subroutine FDH_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by FDH_solve, at the end of the solution process. It is important to do this if the data object is re-used for another problem **with a different structure** since FDH_initialize cannot test for this situation, and any existing associated targets will subsequently become unreachable.

We use square brackets [ ] to indicate OPTIONAL arguments.

2.3.1 The initialization subroutine

Default values are provided as follows:

```
CALL FDH_initialize( data, control, inform )
```

`data` is a scalar `INTENT(INOUT)` argument of type `FDH_data_type` (see Section 2.2.3). It is used to hold data about the problem being solved.

`control` is a scalar `INTENT(OUT)` argument of type `FDH_control_type` (see Section 2.2.1). On exit, `control` contains default values for the components as described in Section 2.2.1. These values should only be changed after calling `FDH_initialize`.

`inform` is a scalar `INTENT(OUT)` argument of type `FDH_inform_type` (see Section 2.2.2). A successful call to `FDH_initialize` is indicated when the component `status` has the value 0. For other return values of `status`, see Section 2.5.

2.3.2 The analysis subroutine

The analysis phase, in which the sparsity pattern of the Hessian is used to generate information that will be used when estimating its values, is called as follows:

```
CALL FDH_analyse( n, nz, ROW, DIAG, data, control, inform )
```

`n` is a scalar `INTENT(IN)` scalar argument of type `INTEGER(ip_)`, that must be set to n the dimension of the Hessian matrix, i.e. the number of variables in the function f. **Restrictions:** `n` > 0.

`nz` is a scalar `INTENT(IN)` scalar argument of type `INTEGER(ip_)`, that must be set to the number of nonzero entries on and below the diagonal of the Hessian matrix. **Restrictions:** `n` $\leq$ `nz` $\leq$ `n * (n+1)/2`.

`ROW` is a scalar `INTENT(INOUT)` rank-one array argument of type `INTEGER(ip_)` and dimension `nz`, that is used to describe the sparsity structure of the Hessian matrix. It must be set so that `ROW(`i`)`, $i = 1, \ldots,$ `nz` contain the row numbers of the successive nonzero elements of the **lower triangular part (including the diagonal)** of the Hessian matrix when scanned column after column in the natural order. The diagonal entry **must preceed the other entries** in each column. The remaining entries may appear in any order. On exit `ROW` will be as input, but will have been altered in the interim. **Restrictions:** $j \leq$ `ROW(`j`)` $\leq n$, $j = 1, \ldots,$ `nz`.

`DIAG` is a scalar `INTENT(IN)` rank-one array argument of type `INTEGER(ip_)` and dimension `n`, that is used to describe the sparsity structure of the Hessian matrix. It must be set so that `DIAG(`i`)` i, $i = 1, \ldots,$ `n` contain the position of the ith diagonal of the matrix in the list held in `ROW`. **Restrictions:** `ROW(DIAG(`i`)`$= i$, $i = 1, \ldots,$ `n`.

`control` is a scalar `INTENT(IN)` argument of type `FDH_control_type` (see Section 2.2.1). Default values may be assigned by calling `FDH_initialize` prior to the first call to `FDH_analyse`.

`inform` is a scalar `INTENT(INOUT)` argument of type `FDH_inform_type` (see Section 2.2.2). A successful call to `FDH_analyse` is indicated when the component `status` has the value 0. For other return values of `status`, see Section 2.5.

`data` is a scalar `INTENT(INOUT)` argument of type `FDH_data_type` (see Section 2.2.3). It is used to hold data about the problem being solved. With the possible exceptions of the components `eval_status` and `U` (see Section 2.4), it must not have been altered **by the user** since the last call to `FDH_initialize`.

2.3.3 The estimation subroutine

The estimation phase, in which the nonzero entries of the Hessian are estimated by finite differences, is called as follows:

```
CALL FDH_estimate( n, nz, ROW, DIAG, X, G, STEPSIZE, H,    &
                   data, control, inform, userdata, eval_G )
```

`n`, `nz`, `ROW` and `DIAG` are `INTENT(IN)` arguments exactly as described and input to `FDH_analyse`, and must not have been changed in the interim.

`X` is a scalar `INTENT(IN)` rank-one array argument of type `REAL(rp_)`, and dimension `n`, that must be set so that `X(`i`)` contains the component x_i, $i = 1,\ldots,$ `n` of the point $\mathbf{x}$ at which the user wishes to estimate $\mathbf{H}(\mathbf{x})$.

`G` is a scalar `INTENT(IN)` rank-one array argument of type `REAL(rp_)`, and dimension `n`, that must be set so that `G(`i`)` contains the component $g_i(\mathbf{x})$, $i = 1,\ldots,$ `n` of the gradient $\mathbf{g}(\mathbf{x})$ of f at the point $\mathbf{x}$ input in `X`.

`STEPSIZE` is a scalar `INTENT(IN)` rank-one array argument of type `REAL(rp_)`, and dimension `n`, that must be set to the stepsizes to be used in the finite difference scheme. One can roughly say that `STEPSIZE(`i`)` is the step used to evaluate the ith column of the Hessian—recommended values are between 10^{-7} and 10^{-3} times the corresponding component of `X`.

`H` is a scalar `INTENT(INOUT)` rank-one array argument of type `REAL(rp_)`, and dimension `nz`, that needs not be set on input, but that will be set to the non-zeros of the Hessian $\mathbf{H}(\mathbf{x})$ in the order defined by the list stored in `ROW`.

`control` is a scalar `INTENT(IN)` argument of type `FDH_control_type` (see Section 2.2.1). Default values may be assigned by calling `FDH_initialize` prior to the first call to `FDH_analyse`.

`inform` is a scalar `INTENT(INOUT)` argument of type `FDH_inform_type` (see Section 2.2.2). A successful call to `FDH_analyse` is indicated when the component `status` has the value 0. For other return values of `status`, see Sections 2.4 and 2.5.

`data` is a scalar `INTENT(INOUT)` argument of type `FDH_data_type` (see Section 2.2.3). It is used to hold data about the problem being solved. With the possible exceptions of the components `eval_status` and `U` (see Section 2.4), it must not have been altered **by the user** since the last call to `FDH_initialize`.

`userdata` is a scalar `INTENT(INOUT)` argument of type `NLPT_userdata_type` whose components may be used to communicate user-supplied data to and from the `OPTIONAL` subroutine `eval_G`, (see Section 2.2.4).

`eval_G` is an `OPTIONAL` user-supplied subroutine whose purpose is to evaluate the value of the gradient of the objective function $\mathbf{g}(\mathbf{x}) = \nabla_x f(\mathbf{x})$ at a given vector $\mathbf{x}$. See Section 2.3.5 for details. If `eval_G` is present, it must be declared `EXTERNAL` in the calling program. If `eval_G` is absent, `GALAHAD_FDH_analyse` will use reverse communication to obtain gradient values (see Section 2.4).

2.3.4 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL FDH_terminate( data, control, inform )
```

`data` is a scalar `INTENT(INOUT)` argument of type `FDH_data_type` exactly as for `FDH_solve`, which must not have been altered **by the user** since the last call to `FDH_initialize`. On exit, array components will have been deallocated.

`control` is a scalar `INTENT(IN)` argument of type `FDH_control_type` exactly as for `FDH_solve`.

inform is a scalar INTENT(OUT) argument of type FDH_inform_type exactly as for FDH_solve. Only the component status will be set on exit, and a successful call to FDH_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.5.

2.3.5 Gradient values via internal evaluation

If the argument eval_G is present when calling GALAHAD_FDH_analyse, the user is expected to provide a subroutine of that name to evaluate the value of the gradient the objective function $\mathbf{g}(\mathbf{x}) = \nabla_x f(\mathbf{x})$. The routine must be specified as

```
SUBROUTINE eval_G( status, X, userdata, G )
```

whose arguments are as follows:

status is a scalar INTENT(OUT) argument of type INTEGER(ip_), that should be set to 0 if the routine has been able to evaluate the gradient of the objective function and to a non-zero value if the evaluation has not been possible.

X is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{x}$.

userdata is a scalar INTENT(INOUT) argument of type NLPT_userdata_type whose components may be used to communicate user-supplied data to and from the subroutine eval_G (see Section 2.2.4).

G is a rank-one INTENT(OUT) argument of type REAL(rp_), whose components should be set to the values of the gradient of the objective function $\mathbf{g}(\mathbf{x}) = \nabla_x f(\mathbf{x})$ evaluated at the vector $\mathbf{x}$ input in X.

2.4 Reverse Communication Information

A positive value of inform%status on exit from FDH_estimate indicates that GALAHAD_FDH_analyse is seeking further information—this will happen if the user has chosen not to evaluate gradient values internally (see Section 2.3.5). The user should compute the required information and re-enter GALAHAD_FDH_analyse with inform%status and all other arguments (except those specifically mentioned below) unchanged.

Possible values of inform%status and the information required are

1. The user should compute the gradient of the objective function $\mathbf{g}(\mathbf{x}) = \nabla_x f(\mathbf{x})$ at the point $\mathbf{x}$ indicated in data%X. The value of the i-th component of the gradient should be set in data%G(i), for $i = 1, \ldots, n$ and data%eval_status should be set to 0. If the user is unable to evaluate a component of $\mathbf{g}(\mathbf{x})$—for instance, if a component of the gradient is undefined at $\mathbf{x}$—the user need not set data%G, but should then set data%eval_status to a non-zero value.

2.5 Warning and error messages

A negative value of inform%status on exit from FDH_analyse, FDH_estimate or FDH_terminate indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

-1. An allocation error occurred. A message indicating the offending array is written on unit control%error, and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

-2. A deallocation error occurred. A message indicating the offending array is written on unit control%error and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

-3. One or more of the restriction $0 < \mathtt{n} \leq \mathtt{nz} \leq \mathtt{n} * (\mathtt{n}+1)/2$ has been violated.

-23. One or more of the restrictions ROW(DIAG(i)) = i, $i = 1, \ldots,$ n, or $j \leq$ ROW(j) $\leq n$, $j = 1, \ldots,$ nz, has been violated. See inform%bad_row for the index of the row involved.

-31. FDH_estimate has been called before FDH_analyse.

2.6 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable control of type FDH_control_type (see Section 2.2.1), by reading an appropriate data specification file using the subroutine FDH_read_specfile. This facility is useful as it allows a user to change FDH control parameters without editing and recompiling programs that call FDH.

A specification file, or specfile, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by FDH_read_specfile must start with a "BEGIN FDH" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by FDH_read_specfile .. )
  BEGIN FDH
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by FDH_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN FDH" and "END" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN FDH SPECIFICATION
```

and

```
END FDH SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN FDH" and "END" delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy manner to "comment out" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of three different types, namely integer, logical or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are "ON", "TRUE", ".TRUE.", "T", "YES", "Y", or "OFF", "NO", "N", "FALSE", ".FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE".

The specification file must be open for input when FDH_read_specfile is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is REWINDed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by FDH_read_specfile.

2.6.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL FDH_read_specfile( control, device )
```

`control` is a scalar `INTENT(INOUT)` argument of type `FDH_control_type` (see Section 2.2.1). Default values should have already been set, perhaps by calling `FDH_initialize`. On exit, individual components of `control` may have been changed according to the commands found in the specfile. Specfile commands and the component (see Section 2.2.1) of `control` that each affects are given in Table 2.1.

command	component of `control`	value type
`error-printout-device`	`%error`	integer
`printout-device`	`%out`	integer
`print-level`	`%print_level`	integer
`space-critical`	`%space_critical`	logical
`deallocate-error-fatal`	`%deallocate_error_fatal`	logical
`output-line-prefix`	`%prefix`	character

Table 2.1: Specfile commands and associated components of `control`.

`device` is a scalar `INTENT(IN)` argument of type `INTEGER(ip_)`, that must be set to the unit number on which the specfile has been opened. If `device` is not open, `control` will not be altered and execution will continue, but an error message will be printed on unit `control%error`.

2.7 Information printed

If `control%print_level` is positive, information about errors encountered will be printed on unit `control%out`.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: `FDH_solve` calls the GALAHAD packages `GALAHAD_SYMBOLS`, `GALAHAD_SPECFILE`, `GALAHAD_SPACE` and `GALAHAD_NLPT`.

Input/output: Output is under control of the arguments `control%error`, `control%out` and `control%print_level`.

Restrictions: $0 < \mathtt{n} \leq \mathtt{nz} \leq \mathtt{n} * (\mathtt{n}+1)/2$, $\mathtt{ROW}(\mathtt{DIAG}(i)) = i$, $i = 1, \ldots, \mathtt{n}$, or $j \leq \mathtt{ROW}(j) \leq n$, $j = 1, \ldots, \mathtt{nz}$.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

4 METHOD

The routines use a "Lower triangular substitution algorithm". It assumes that no diagonal values are constrained by the sparsity requirements. The analysis phase uses the sparsity pattern of the matrix to decide how many differences in gradient are needed for estimation and along what directions. For this purpose, it defines a symmetric permutation of

the matrix. The evaluation phase computes the differences in gradients that are required and then solves a consequent linear system by a substitution to obtain the entries of the approximate Hessian.

Once the pattern analysis is performed, one can approximate the Hessian of f at several different points. This is done by a single call to `FDH_analyse` followed by several calls to `FDH_estimate` for different values of x with corresponding $g(x)$.

Reference:

The method is described in detail in

M. J. D. Powell and Ph. L. Toint. "On the estimation of sparse Hessian matrices". SIAM J. Numer. Anal. **16** (1979) 1060-1074.

5 EXAMPLES OF USE

Suppose we wish to estimate the Hessian matrix of the objective function

$$f(x_1,\ldots,x_5) = (x_1+p)^3 + x_2^3 + x_3^3 + x_4^3 + x_5^3 + x_1x_4 + x_2x_3 + x_3x_4 + x_4x_5,$$

that depends on the parameter p, whose gradient is

$$\mathbf{g}(\mathbf{x}) = \begin{pmatrix} 3(x_1+p)^2 + x_4 \\ 3x_2^2 + x_3 \\ 3x_3^2 + x_2 + x_4 \\ 3x_4^2 + x_1 + x_3 + x_5 \\ 3x_5^2 + x_4 \end{pmatrix}$$

and thus whose Hessian has the sparsity pattern

$$\begin{pmatrix} * & 0 & 0 & \cdot & 0 \\ 0 & * & \cdot & 0 & 0 \\ 0 & * & * & \cdot & 0 \\ * & 0 & * & * & \cdot \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

(the entries $*$ in the lower triangle are the ones that will be estimated).

Choosing uniform step lengths of size 10^{-6}, we may estimate the Hessian at $x = (1,1,1,1,1)^T$ and $(1,2,3,4,5)^T$ for $p = 4$ using the following code - we use an explicit call to evaluate the gradient at the first point and the reverse-communication method at the second:

```
! THIS VERSION: GALAHAD 4.1 - 2022-12-19 AT 14:50 GMT.
   PROGRAM GALAHAD_FDH_EXAMPLE
   USE GALAHAD_FDH_double         ! double precision version
   IMPLICIT NONE
   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
   TYPE ( FDH_data_type ) :: data
   TYPE ( FDH_control_type ) :: control
   TYPE ( FDH_inform_type ) :: inform
   INTEGER, PARAMETER :: n = 5, nz = 9
   REAL ( KIND = wp ), PARAMETER :: p = 4.0_wp
   INTEGER :: i, status
   INTEGER :: DIAG( n ), ROW( nz )
   REAL ( KIND = wp ) ::  X1( n ), X2( n ), STEPSIZE( n ), G( n )
```

```
   REAL ( KIND = wp ) ::  H( nz )
   TYPE ( GALAHAD_userdata_type ) :: userdata
   INTERFACE
     SUBROUTINE GRAD( status, X, userdata, G )
     USE GALAHAD_USERDATA_double, ONLY: GALAHAD_userdata_type
     INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
     INTEGER, INTENT( OUT ) :: status
     REAL ( KIND = wp ), DIMENSION( : ), INTENT( IN ) :: X
     REAL ( KIND = wp ), DIMENSION( : ), INTENT( OUT ) :: G
     TYPE ( GALAHAD_userdata_type ), INTENT( INOUT ) :: userdata
     END SUBROUTINE GRAD
   END INTERFACE
! start problem data
   ROW = (/ 1, 4, 2, 3, 3, 4, 4, 5, 5 /)      ! Record the sparsity pattern of
   DIAG = (/ 1, 3, 5, 7, 9 /)                 ! the Hessian (lower triangle)
   ALLOCATE( userdata%real( 1 ) )             ! Allocate space for the parameter
   userdata%real( 1 ) = p                     ! Record the parameter, p
   STEPSIZE  = (/ 0.000001_wp, 0.000001_wp, 0.000001_wp,                      &
                  0.000001_wp, 0.000001_wp /) ! Set difference stepsizes
! estimate the Hessian at X1 by internal evaluation
   X1 = (/ 1.0_wp, 1.0_wp, 1.0_wp, 1.0_wp, 1.0_wp /)
   CALL FDH_initialize( data, control, inform )
   CALL FDH_analyse( n, nz, ROW, DIAG, data, control, inform )
   IF ( inform%status /= 0 ) THEN             ! Failure
     WRITE( 6, "( ' return with nonzero status ', I0, ' from FDH_analyse' )" ) &
       inform%status ; STOP
   END IF
   CALL GRAD( status, X1( : n ), userdata, G( : n ) )
   CALL FDH_estimate( n, nz, ROW, DIAG, X1, G, STEPSIZE, H,                   &
                      data, control, inform, userdata, eval_G = GRAD )
   IF ( inform%status == 0 ) THEN              ! Success
     WRITE( 6, "( ' At 1st point, nonzeros in Hessian matrix are ', /,        &
    &             ( 5ES12.4 ) )" ) ( H( i ), i = 1, nz )
   ELSE                                        ! Failure
     WRITE( 6, "( ' return with nonzero status ', I0, ' from FDH_estimate' )" )&
       inform%status ; STOP
   END IF
! estimate the Hessian at X2 by reverse communication
   X2 = (/ 1.0_wp, 2.0_wp, 3.0_wp, 4.0_wp, 5.0_wp /)
   CALL GRAD( status, X2( : n ), userdata, G( : n ) )
10 CONTINUE
   CALL FDH_estimate( n, nz, ROW, DIAG, X2, G, STEPSIZE, H,                   &
                      data, control, inform, userdata )
   IF ( inform%status == 0 ) THEN              ! Success
     WRITE( 6, "( /, ' At 2nd point, nonzeros in Hessian matrix are ', /,     &
    &             ( 5ES12.4 ) )" ) ( H( i ), i = 1, nz )
   ELSE IF ( inform%status > 0 ) THEN          ! Reverse communication required
     CALL GRAD( data%eval_status, data%X( : n ), userdata, data%G( : n ) )
     GO TO 10
   ELSE                                        ! Failure
     WRITE( 6, "( ' return with nonzero status ', I0, ' from FDH_estimate' )" )&
       inform%status ; STOP
   END IF
   CALL FDH_terminate( data, control, inform ) ! Delete internal workspace
   END PROGRAM GALAHAD_FDH_EXAMPLE
```

```
! internal subroutine to evaluate the gradient of the objective
   SUBROUTINE GRAD( status, X, userdata, G )
   USE GALAHAD_USERDATA_double, ONLY: GALAHAD_userdata_type
   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
   INTEGER, INTENT( OUT ) :: status
   REAL ( KIND = wp ), DIMENSION( : ), INTENT( IN ) :: X
   REAL ( KIND = wp ), DIMENSION( : ), INTENT( OUT ) :: G
   TYPE ( GALAHAD_userdata_type ), INTENT( INOUT ) :: userdata
   G( 1 ) = 3.0_wp * ( X( 1 ) + userdata%real( 1 ) ) ** 2 + X( 4 )
   G( 2 ) = 3.0_wp * X( 2 ) ** 2 + X( 3 )
   G( 3 ) = 3.0_wp * X( 3 ) ** 2 + X( 2 ) + X( 4 )
   G( 4 ) = 3.0_wp * X( 4 ) ** 2 + X( 1 ) + X( 3 ) + X( 5 )
   G( 5 ) = 3.0_wp * X( 5 ) ** 2 + X( 4 )
   status = 0
   END SUBROUTINE GRAD
```

Notice how the parameter p is passed to the function evaluation routines via the `real` component of the derived type `userdata`. The code produces the following output:

```
 At 1st point, nonzeros in Hessian matrix are
  3.0000E+01  1.0000E+00  6.0000E+00  1.0000E+00  6.0000E+00
  1.0000E+00  6.0000E+00  1.0000E+00  6.0000E+00

 At 2nd point, nonzeros in Hessian matrix are
  3.0000E+01  1.0000E+00  1.2000E+01  1.0000E+00  1.8000E+01
  1.0000E+00  2.4000E+01  1.0000E+00  3.0000E+01
```