



---

# GALAHAD

# FDC

---

PACKAGE SPECIFICATION

GALAHAD Optimization Library version 2.2

---

## 1 SUMMARY

Given an under-determined set of linear equations/constraints  $\mathbf{a}_i^T \mathbf{x} = c_i$ ,  $i = 1, \dots, m$  involving  $n \geq m$  unknowns  $\mathbf{x}$ , this package **determines whether the constraints are consistent, and if so how many of the constraints are dependent**; a list of dependent constraints, that is, those which may be removed without changing the solution set, will be found and the remaining  $\mathbf{a}_i$  will be linearly independent. Full advantage is taken of any zero coefficients in the vectors  $\mathbf{a}_i$ .

**ATTRIBUTES — Versions:** GALAHAD\_FDC\_single, GALAHAD\_FDC\_double. **Uses:** GALAHAD\_SYMBOLS, GALAHAD\_SMT, GALAHAD\_ROOTS, GALAHAD\_SILS, GALAHAD\_SPECFILE, GALAHAD\_SPACE. **Date:** August 2006. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

## 2 HOW TO USE THE PACKAGE

Access to the package requires a USE statement such as

*Single precision version*

```
USE GALAHAD_FDC_single
```

*Double precision version*

```
USE GALAHAD_FDC_double
```

If it is required to use both modules at the same time, the derived types SMT\_type, FDC\_time\_type, FDC\_control\_type, and FDC\_inform\_type (Section 2.1) and the subroutines FDC\_initialize, FDC\_find\_dependent, FDC\_terminate (Section 2.2) and FDC\_read\_specfile (Section 2.4) must be renamed on one of the USE statements.

### 2.1 The derived data types

Three derived data types are accessible from the package.

#### 2.1.1 The derived data type for holding control parameters

The derived data type FDC\_control\_type is used to hold controlling data. Default values may be obtained by calling FDC\_initialize (see Section 2.2.1), while components may also be changed by calling GALAHAD\_FDC\_read\_spec (see Section 2.4.1). The components of FDC\_control\_type are:

**error** is a scalar variable of type default INTEGER, that holds the stream number for error messages. Printing of error messages in FDC\_find\_dependent and FDC\_terminate is suppressed if  $\text{error} \leq 0$ . The default is  $\text{error} = 6$ .

**out** is a scalar variable of type default INTEGER, that holds the stream number for informational messages. Printing of informational messages in FDC\_find\_dependent is suppressed if  $\text{out} < 0$ . The default is  $\text{out} = 6$ .

**print\_level** is a scalar variable of type default INTEGER, that is used to control the amount of informational output which is required. No informational output will occur if  $\text{print\_level} \leq 0$ . If  $\text{print\_level} = 1$ , basic statistics of the performance of the package will be produced. If  $\text{print\_level} \geq 2$ , this output will be increased to provide details such as the size of each neglected pivot. The default is  $\text{print\_level} = 0$ .

---

**All use is subject to licence. See <http://galahad.rl.ac.uk/galahad-www/cou.html>.**  
**For any commercial application, a separate license must be signed.**

`indmin` is a scalar variable of type default INTEGER, that specifies an initial estimate as to the amount of integer workspace required by the factorization package SILS. The default is `indmin = 1000`.

`valmin` is a scalar variable of type default INTEGER, that specifies an initial estimate as to the amount of real workspace required by the factorization package SILS. The default is `valmin = 1000`.

`max_infeas` is a scalar variable of type default REAL (double precision in GALAHAD\_FDC\_double), that holds the largest permitted infeasibility for a dependent constraint. Specifically, if  $\mathbf{x}$  satisfies  $\mathbf{a}_i^T \mathbf{x} = c_i$  for the constraints deemed to be linearly independent, it is required that  $|\mathbf{a}_i^T \mathbf{x} - c_i| \leq \text{max\_infeas}$  for those classified as dependent. The default is `max_infeas =  $u^{1/3}$` , where  $u$  is `EPSILON(1.0)` (`EPSILON(1.0D0)` in GALAHAD\_FDC\_double).

`pivot_tol` is a scalar variable of type default REAL (double precision in GALAHAD\_FDC\_double), that holds the threshold pivot tolerance used by the matrix factorization when attempting to detect linearly dependent constraints. A value larger than `pivot_tol` is appropriate. See the documentation for the package SILS for details. The default is `pivot_tol = 0.5`.

`zero_pivot` is a scalar variable of type default REAL (double precision in GALAHAD\_FDC\_double). Any pivots smaller than `zero_pivot` in absolute value will be regarded to be zero when attempting to detect linearly dependent constraints. The default is `zero_pivot =  $u^{3/4}$` , where  $u$  is `EPSILON(1.0)` (`EPSILON(1.0D0)` in GALAHAD\_FDC\_double).

`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 ignoring 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 = ""`.

`CNTL` is a scalar variable of type `SILS_control` that is used to control various aspects of the factorization package SILS. See the documentation for GALAHAD\_SILS for more details.

### 2.1.2 The derived data type for holding timing information

The derived data type `FDC_time_type` is used to hold elapsed CPU times for the various parts of the calculation. The components of `FDC_time_type` are:

`total` is a scalar variable of type default REAL, that gives the total time spent in the package.

`analyse` is a scalar variable of type default REAL, that gives the time spent analysing the required matrices prior to factorization.

`factorize` is a scalar variable of type default REAL, that gives the time spent factorizing the required matrices.

---

**All use is subject to licence. See <http://galahad.rl.ac.uk/galahad-www/cou.html> .  
For any commercial application, a separate license must be signed.**

### 2.1.3 The derived data type for holding informational parameters

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

`status` is a scalar variable of type default `INTEGER`, that gives the exit status of the algorithm. See Section 2.3 for details.

`alloc_status` is a scalar variable of type default `INTEGER`, 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`.

`factorization_status` is a scalar variable of type default `INTEGER`, that gives the return status from the matrix factorization.

`factorization_integer` is a scalar variable of type default `INTEGER`, that gives the amount of integer storage used for the matrix factorization.

`factorization_real` is a scalar variable of type default `INTEGER`, that gives the amount of real storage used for the matrix factorization.

`non_negligible_pivot` is a scalar variable of type default `REAL` (double precision in `GALAHAD_FDC_double`), that holds the value of the smallest pivot larger than `control%zero_pivot` when searching for dependent linear constraints. If `non_negligible_pivot` is close to `control%zero_pivot`, this may indicate that there are further dependent constraints, and it may be worth increasing `control%zero_pivot` above `non_negligible_pivot` and solving again.

`time` is a scalar variable of type `FDC_time_type` whose components are used to hold elapsed CPU times for the various parts of the calculation (see Section 2.1.2).

## 2.2 Argument lists and calling sequences

There are three procedures for user calls (see Section 2.4 for further features):

1. The subroutine `FDC_initialize` is used to set default values before attempting to identify dependent constraints.
2. The subroutine `FDC_find_dependent` is called to identify dependent constraints.
3. The subroutine `FDC_terminate` is provided to allow the user to automatically deallocate workspace array components, previously allocated by `FDC_find_dependent`, after use.

### 2.2.1 The initialization subroutine

Default values are provided as follows:

```
CALL FDC_initialize( control, [ FACTORS ] )
```

`control` is a scalar `INTENT(OUT)` argument of type `FDC_control_type` (see Section 2.1.1). On exit, `control` contains default values for the components as described in Section 2.1.1. These values should only be changed after calling `FDC_initialize`.

`FACTORS` is an optional scalar argument of type `SILS_factors`, that will be used as workspace and which must be `PRESENT` if `FACTORS` has not been initialized elsewhere.

---

**All use is subject to licence. See <http://galahad.rl.ac.uk/galahad-www/cou.html>.**  
**For any commercial application, a separate license must be signed.**

### 2.2.2 The subroutine for finding dependent constraints

Dependent constraints are identified as follows:

```
CALL FDC_find_dependent( n, m, A_val, A_col, A_ptr, C, K, FACTORS, &
                        n_depen, C_depen, control, inform )
```

$m$  is a scalar INTENT(IN) argument of type default INTEGER, that holds the number of constraints,  $m$ . **Restriction:**  $m \geq 0$ .

$n$  is a scalar INTENT(IN) argument of type default INTEGER, that holds the number of unknowns,  $n$ . **Restriction:**  $n \geq 0$ .

$A\_val$  is an INTENT(IN) rank-one array of type default REAL (double precision in GALAHAD\_FDC\_double), that holds the values of the entries (that is those component whose values are nonzero) of the matrix  $\mathbf{A}$  whose rows are the vectors  $\mathbf{a}_i^T, i = 1, \dots, m$ . The entries for row  $i$  must directly precede those in row  $i$ , but the order within each row is unimportant.

$A\_col$  is an INTENT(IN) rank-one array of type default INTEGER, that holds the (column) indices of the entries of  $\mathbf{A}$  corresponding to the values input in  $A\_val$ .

$A\_ptr$  is an INTENT(IN) rank-one array of dimension  $m+1$  and type default INTEGER, whose  $i$ -th entry holds the starting position of row  $i$  of  $\mathbf{A}$  for  $i = 1, \dots, m$ . The  $m+1$ -st entry of  $A\_ptr$  must hold the total number of entries plus one.

$C$  is an INTENT(IN) rank-one array of dimension  $m$  and type default REAL (double precision in GALAHAD\_FDC\_double), whose  $i$ -th component must be set to  $c_i$  for  $i = 1, \dots, m$ .

$K$  is a scalar INTENT(INOUT) argument of type SMT\_type that need not be set on entry and will be used as workspace.

$FACTORS$  is a scalar INTENT(INOUT) argument of type SILS\_factors that need not be set on entry and will be used as workspace.

$n\_depen$  is a scalar INTENT(OUT) argument of type default INTEGER, that gives the number of dependent constraints.

$C\_depen$  is a rank-one allocatable array of type default INTEGER, that should either have been nullified or associated on entry, but need not be set. On exit, if  $n\_depen > 0$ , it will have been allocated to be of length  $n\_depen$  and its components will be the indices of the dependent constraints. It will not be allocated or set if  $n\_depen = 0$ .

$control$  is a scalar INTENT(IN) argument of type FDC\_control\_type (see Section 2.1.1). Default values may be assigned by calling FDC\_initialize prior to the first call to FDC\_find\_dependent.

$inform$  is a scalar INTENT(OUT) argument of type FDC\_inform\_type (see Section 2.1.3). A successful call to FDC\_find\_dependent is indicated when the component status has the value 0. For other return values of status, see Section 2.3.

### 2.2.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL FDC_terminate( K, FACTORS, C_depen, control, inform )
```

$K$  is a scalar INTENT(INOUT) argument of type SMT\_type whose array components will be deallocated on exit.

$FACTORS$  is a scalar INTENT(INOUT) argument of type SILS\_factors whose array components will be deallocated on exit.

---

**All use is subject to licence. See <http://galahad.rl.ac.uk/galahad-www/cou.html>.**  
**For any commercial application, a separate license must be signed.**

`C_depen` is a rank-one allocatable array of type default `INTEGER`, that will be deallocated on exit.

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

`inform` is a scalar `INTENT(OUT)` argument of type `FDC_inform_type` exactly as for `FDC_find_dependent`. Only the components `status` `alloc_status` and `bad_alloc` will be set on exit, and a successful call to `FDC_terminate` is indicated when this component `status` has the value 0. For other return values of `status`, see Section 2.3.

### 2.3 Warning and error messages

A negative value of `inform%status` on exit from `FDC_find_dependent` or `FDC_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. `status` is given by the value `inform%alloc_status`.
- 3. One of the restrictions  $n \geq 0$  or  $m \geq 0$  has been violated.
- 5 The constraints appear to be inconsistent.
- 9. The analysis phase of the factorization failed; the return status from the factorization package is given in the component `inform%factor_status`.
- 10. The factorization failed; the return status from the factorization package is given in the component `inform%factor_status`.

### 2.4 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable `control` of type `GALAHAD_FDC_control_type` (see Section 2.1.1), by reading an appropriate data specification file using the subroutine `GALAHAD_FDC_read_specfile`. This facility is useful as it allows a user to change FDC control parameters without editing and recompiling programs that call FDC.

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 `GALAHAD_FDC_read_specfile` must start with a "BEGIN GALAHAD\_FDC" command and ends with an "END" command. The syntax of the `specfile` is thus defined as follows:

```
( .. lines ignored by FDC_read_specfile .. )
  BEGIN FDC
    keyword    value
    .....
    keyword    value
  END
( .. lines ignored by FDC_read_specfile .. )
```

---

**All use is subject to licence. See <http://galahad.rl.ac.uk/galahad-www/cou.html>.**  
**For any commercial application, a separate license must be signed.**

where keyword and value are two strings separated by (at least) one blank. The “BEGIN GALAHAD\_FDC” 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 FDC SPECIFICATION
```

and

```
END FDC SPECIFICATION
```

are acceptable. Furthermore, between the “BEGIN GALAHAD\_FDC” 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 GALAHAD\_FDC\_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 GALAHAD\_FDC\_read\_specfile.

#### 2.4.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL FDC_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type GALAHAD\_FDC\_control\_type (see Section 2.1.1). Default values should have already have been set, perhaps by calling GALAHAD\_FDC\_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 2.1.1) 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
initial-integer-workspace	%indmin	integer
initial-real-workspace	%valmin	integer
maximum-permitted-infeasibility	%max_infeas	real
pivot-tolerance-used-for-dependencies	%pivot_tol	real
zero-pivot-tolerance	%zero_pivot	real
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical

Table 2.1: Specfile commands and associated components of control.

device is a scalar INTENT(IN) argument of type default INTEGER, 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.

---

**All use is subject to licence. See <http://galahad.rl.ac.uk/galahad-www/cou.html>.**  
**For any commercial application, a separate license must be signed.**

## 2.5 Information printed

If `control%print_level` is positive, information about the progress of the algorithm will be printed on unit `control%out`. If `control%print_level = 1`, basic statistics of the performance of the package will be produced. If `control%print_level ≥ 2` this output will be increased to provide details such as the size of each neglected pivot.

## 3 GENERAL INFORMATION

**Use of common:** None.

**Workspace:** Provided automatically by the module.

**Other routines called directly:** None.

**Other modules used directly:** `FDC_find_dependent` calls the GALAHAD packages `GALAHAD_SYMBOLS`, `GALAHAD_SMT`, `GALAHAD_ROOTS`, `GALAHAD_SILS`, `GALAHAD_SPECFILE` and `GALAHAD_SPACE`.

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

**Restrictions:**  $n ≥ 0, m ≥ 0$ .

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

## 4 METHOD

The matrix

$$\mathbf{K} = \begin{pmatrix} \alpha \mathbf{I} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{pmatrix}$$

is formed and factorized for some small  $\alpha > 0$  using the GALAHAD package `SILS`—the factors  $\mathbf{K} = \mathbf{P}\mathbf{L}\mathbf{D}\mathbf{L}^T\mathbf{P}^T$  are used to determine whether  $\mathbf{A}$  has dependent rows. In particular, in exact arithmetic dependencies in  $\mathbf{A}$  will correspond to zero pivots in the block diagonal matrix  $\mathbf{D}$ . The factorization may also be used to determine whether the system is consistent.

## 5 EXAMPLE OF USE

Suppose we wish to find whether the linear constraints  $x_1 + 2x_2 + 3x_3 + 4x_4 = 5$ ,  $2x_1 - 4x_2 + 6x_3 - 8x_4 = 10$  and  $5x_2 + 10x_4 = 0$  are consistent but redundant. Then we may use the following code.

```
! THIS VERSION: GALAHAD 2.1 - 22/03/2007 AT 09:00 GMT.
PROGRAM GALAHAD_FDC_example
USE GALAHAD_FDC_double                      ! double precision version
IMPLICIT NONE
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
INTEGER, PARAMETER :: n = 4, m = 3, a_ne = 10
INTEGER :: A_ptr( m + 1 ), A_col( a_ne )
REAL ( KIND = wp ) :: A_val( a_ne ), C( m )
INTEGER, ALLOCATABLE :: C_depen( : ) ! Remeber to nullify C_depen
INTEGER :: n_depen
TYPE ( SMT_type ) :: K
TYPE ( SILS_factors ) :: FACTORS
TYPE ( FDC_control_type ) :: control
TYPE ( FDC_inform_type ) :: inform
```

---

**All use is subject to licence. See <http://galahad.rl.ac.uk/galahad-www/cou.html>.**  
**For any commercial application, a separate license must be signed.**

```
A_val = (/ 1.0_wp, 2.0_wp, 3.0_wp, 4.0_wp, 2.0_wp, -4.0_wp, 6.0_wp,      &
          -8.0_wp, 5.0_wp, 10.0_wp /)
A_col = (/ 1, 2, 3, 4, 1, 2, 3, 4, 2, 4 /)
A_ptr = (/ 1, 5, 9, 11 /)
C = (/ 5.0_wp, 10.0_wp, 0.0_wp /)
CALL FDC_initialize( control, FACTORS )      ! Initialize control parameters
CALL FDC_find_dependent( n, m, A_val, A_col, A_ptr, C, K, FACTORS, n_depen, &
  C_depen, control, inform )                ! Check for dependencies
IF ( inform%status == 0 ) THEN              ! Successful return
  IF ( n_depen == 0 ) THEN
    WRITE( 6, "( ' FDC_find_dependent - no dependencies ' )" )
  ELSE
    WRITE( 6, "( ' FDC_find_dependent - dependent constraint(s):', 3I3 )" ) &
      C_depen
  END IF
ELSE
  ! Error returns
  WRITE( 6, "( ' FDC_find_dependent exit status = ', I6 ) " ) inform%status
END IF
CALL FDC_terminate( K, FACTORS, C_depen, control, inform ) ! Delete workspace
END PROGRAM GALAHAD_FDC_example
```

This produces the following output:

```
FDC_find_dependent - dependent constraint(s): 1
```