# 1 SUMMARY

Given a sparse symmetric matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$ and the factorization of $\mathbf{A}$ found by the GALAHAD package GALAHAD_SLS, this package **solves the system of linear equations $\mathbf{Ax} = \mathbf{b}$ using iterative refinement.**

**ATTRIBUTES — Versions:** GALAHAD_IR_single, GALAHAD_IR_double. **Uses:** GALAHAD_SYMBOLS, GALAHAD_SPACE, GALAHAD_SMT, GALAHAD_QPT, GALAHAD_SLS, GALAHAD_SPECFILE. **Date:** October 2008. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory **Language:** Fortran 95 + TR 15581 or Fortran 2003.

# 2 HOW TO USE THE PACKAGE

## 2.1 Calling sequences

Access to the package requires a USE statement such as

*Single precision version*

```
USE GALAHAD_IR_single
```

*Double precision version*

```
USE GALAHAD_IR_double
```

If it is required to use both modules at the same time, the derived types SMT_type, IR_control_type, IR_inform_type, IR_data_type, SLS_factors, (Section 2.2) and the subroutines IR_initialize, IR_solve, IR_terminate (Section 2.3) and IR_read_specfile (Section 2.5) must be renamed on one of the USE statements.

## 2.2 The derived data types

Five derived data types are accessible from the package.

### 2.2.1 The derived data type for holding the matrix

The derived data type SMT_type is used to hold the matrix $\mathbf{A}$. The components of SMT_type are:

n     is a scalar variable of type default INTEGER, that holds the order $n$ of the matrix $\mathbf{A}$. **Restriction:** n $\geq$ 1.

ne    is a scalar variable of type default INTEGER, that holds the number of matrix entries. **Restriction:** ne $\geq$ 0.

VAL   is a rank-one allocatable array of type default REAL (double precision in GALAHAD_IR_double), and dimension at least ne, that holds the values of the entries. Each pair of off-diagonal entries $a_{ij} = a_{ji}$ is represented as a single entry. Duplicated entries are summed.

ROW   is a rank-one allocatable array of type default INTEGER, and dimension at least ne, that holds the row indices of the entries.

COL   is a rank-one allocatable array of type default INTEGER, and dimension at least ne, that holds the column indices of the entries.

**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 derived data type for holding control parameters

The derived data type `IR_control_type` is used to hold controlling data. Default values may be obtained by calling `IR_initialize` (see Section 2.3.1). The components of `IR_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 `IR_solve` and `IR_terminate` is suppressed if $\text{error} \leq 0$. The default is `error = 6`.

`out` is a scalar variable of type default `INTEGER`, that holds the stream number for informational messages. Printing of informational messages in `IR_solve` is suppressed if $\text{out} < 0$. The default is `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 `print_level = 1` a single line of output will be produced for each iteration of the process. If $\text{print\_level} \geq 2$ this output will be increased to provide significant detail of each iteration. The default is `print_level = 0`.

`itref_max` is a scalar variable of type default `INTEGER`, that holds the maximum number of iterative refinements which will be allowed. The default is `itref_max = 1`.

`acceptable_residual_relative` and `acceptable_residual_absolute` are scalar variables of type `REAL` (double precision in `GALAHAD_IR_double`), that specify an acceptable level for the residual $\mathbf{Ax} - \mathbf{b}$. In particular, iterative refinement will cease as soon as $\|\mathbf{Ax} - \mathbf{b}\|_\infty$ falls below $\max(\|\mathbf{b}\|_\infty * \text{acceptable\_residual\_relative},$ `acceptable_residual_absolute`). The defaults are `acceptable_residual_relative` = `acceptable_residual_absolute` = $10u$, where $u$ is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_IR_double`).

`space_critical` is a scalar variable of type default `LOGICAL`, that may be set `.TRUE.` if the user wishes the package to allocate as little internal storage as possible, and `.FALSE.` otherwise. The package may be more efficient if `space_critical` is set `.FALSE.`. The default is `space_critical = .FALSE.`.

`deallocate_error_fatal` is a scalar variable of type default `LOGICAL`, that may be set `.TRUE.` if the user wishes the package to return to the user in the unlikely event that an internal array deallocation fails, and `.FALSE.` if the package should be allowed to try to continue. 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 = ""`.

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

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

`status` is a scalar variable of type default `INTEGER`, that gives the current status of the algorithm. See Section 2.4 for details.

`alloc_status` is a scalar variable of type default `INTEGER`, that gives the status of the last internal 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`.

---

### 2.2.4 The derived data type for holding problem data

The derived data type `IR_data_type` is used to hold all the data for a particular problem between calls of `IR` procedures. This data should be preserved, untouched, from the initial call to `IR_initialize` to the final call to `IR_terminate`.

### 2.2.5 The derived data type for holding factors of a matrix

The derived data type `SLS_FACTORS` is used to hold the factors and related data for a matrix. All components are private.

### 2.3 Argument lists and calling sequences

There are three procedures for user calls (see Section 2.5 for further features):

1. The subroutine `IR_initialize` is used to set default values, and initialize private data.

2. The subroutine `IR_solve` is called to solve $\mathbf{Ax} = \mathbf{b}$; this must have been preceded by a call to `SLS_factorize` to obtain the factors of $\mathbf{A}$.

3. The subroutine `IR_terminate` is provided to allow the user to automatically deallocate array components of the private data, allocated by `IR_solve`, at the end of the solution process.

### 2.3.1 The initialization subroutine

Default values are provided as follows:

        CALL IR_initialize( data, control, inform )

data   is a scalar INTENT(INOUT) argument of type `IR_data_type` (see Section 2.2.4). It is used to hold data about the problem being solved.

control   is a scalar INTENT(OUT) argument of type `IR_control_type` (see Section 2.2.2). On exit, control contains default values for the components as described in Section 2.2.2. These values should only be changed after calling `IR_initialize`.

inform   is a scalar INTENT(OUT) argument of type `IR_inform_type` (see Section 2.2.3). A successful call to `IR_initialize` is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

### 2.3.2 The iterative refinement subroutine

The iterative refinement algorithm is called as follows:

        CALL IR_solve( A, X, data, SLS_data, control, SLS_control, inform, SLS_inform )

A   is scalar, of INTENT(IN) and of type SMT_TYPE that holds the matrix $\mathbf{A}$. All components must be unaltered since the call to `SLS_factorize`.

X   is an array INTENT(INOUT) argument of dimension A%n and type default REAL (double precision in GALAHAD_-IR_double), that must be set on input to contain the vector $\mathbf{b}$. On exit, X holds an estimate of the solution $\mathbf{x}$

data   is a scalar INTENT(INOUT) argument of type `IR_data_type` (see Section 2.2.4). It is used to hold data about the problem being solved. It must not have been altered **by the user** since the last call to `IR_initialize`.

---

SLS_data is scalar, of INTENT(INOUT) and of type SLS_data_type that holds the factors of **A** and related data. All components must be unaltered since the call to SLS_factorize.

control is a scalar INTENT(IN) argument of type IR_control_type. (see Section 2.2.2). Default values may be assigned by calling IR_initialize prior to the first call to IR_solve.

SLS_control is a scalar INTENT(IN) argument of type SLS_control_type that is used to control various aspects of the external packages used to solve the symmetric linear systems that arise. See the documentation for the GALAHAD package SLS for further details. All components must be unaltered since the call to SLS_factorize.

inform is a scalar INTENT(INOUT)argument of type IR_inform_type (see Section 2.2.3). A successful call to IR_solve is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

SLS_inform is a scalar INTENT(INOUT)argument of type SLS_inform_type that is used to pass information concerning the progress of the external packages used to solve the symmetric linear systems that arise. See the documentation for the GALAHAD package SLS for further details.

### 2.3.3   The termination subroutine

All previously allocated arrays are deallocated as follows:

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

data is a scalar INTENT(INOUT)argument of type IR_data_type exactly as for IR_solve that must not have been altered **by the user** since the last call to IR_initialize. On exit, array components will have been deallocated.

control is a scalar INTENT(IN)argument of type IR_control_type exactly as for IR_solve.

inform is a scalar INTENT(OUT)argument of type IR_inform_type exactly as for IR_solve. Only the component status will be set on exit, and a successful call to IR_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.4.

### 2.4   Warning and error messages

A negative value of inform%status on exit from IR_solve or IR_terminate indicates that an error might have 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.

### 2.5   Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable control of type IR_control_type (see Section 2.2.2), by reading an appropriate data specification file using the subroutine IR_read_specfile. This facility is useful as it allows a user to change IR control parameters without editing and recompiling programs that call IR.

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 `IR_read_specfile` must start with a "`BEGIN IR`" command and end with an "`END`" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by IR_read_specfile .. )
  BEGIN IR
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by IR_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "`BEGIN IR`" 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 IR SPECIFICATION
```

and

```
END IR SPECIFICATION
```

are acceptable. Furthermore, between the "`BEGIN IR`" 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 `IR_read_specfile` is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is `REWIND`ed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by `IR_read_specfile`.

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

Control parameters may be read from a file as follows:

```
CALL IR_read_specfile( control, device )
```

control is a scalar `INTENT(INOUT)`argument of type `IR_control_type` (see Section 2.2.2). Default values should have already been set, perhaps by calling `IR_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 2.2.2) of control that each affects are given in Table 2.1.

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.

---

| command | component of `control` | value type |
|---|---|---|
| `error-printout-device` | `%error` | integer |
| `printout-device` | `%out` | integer |
| `print-level` | `%print_level` | integer |
| `maximum-refinements` | `%itref_max` | integer |
| `acceptable-residual-relative` | `%acceptable_residual_relative` | real |
| `acceptable-residual-absolute` | `%acceptable_residual_absolute` | real |
| `space-critical` | `%space_critical` | logical |
| `deallocate-error-fatal` | `%deallocate_error_fatal` | logical |

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

### 2.6 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`, the final value of the norm of the residual will be given. If `control%print_level > 1`, the norm of the residual at each iteration will be printed.

## 3 GENERAL INFORMATION

**Use of common:** None.

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

**Other routines called directly:** None.

**Other modules used directly:** `IR_solve` calls the GALAHAD packages `GALAHAD_SYMBOLS`, `GALAHAD_SPACE`, `GALA-HAD_SMT`, `GALAHAD_QPT`, `GALAHAD_SLS`, and `GALAHAD_SPECFILE`.

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

**Restrictions:** None.

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

## 4 METHOD

Iterative refinement proceeds as follows. First obtain the floating-point solution to $\mathbf{Ax} = \mathbf{b}$ using the factors of $\mathbf{A}$. Then iterate until either the desired residual accuracy (or the iteration limit is reached) as follows: evaluate the residual $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$, find the floating-point solution $\delta\mathbf{x}$ to $\mathbf{A}\delta\mathbf{x} = \mathbf{r}$, and replace $\mathbf{x}$ by $\mathbf{x} + \delta\mathbf{x}$.

## 5 EXAMPLE OF USE

Suppose we wish to solve the set of equations

$$\begin{pmatrix} 2 & 3 & & & \\ 3 & & 4 & & 6 \\ & 4 & 1 & 5 & \\ & & 5 & & \\ & 6 & & & 1 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 8 \\ 45 \\ 31 \\ 15 \\ 17 \end{pmatrix}$$

Then we may use the following code

**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.**

```
    PROGRAM GALAHAD_IR_EXAMPLE  !  GALAHAD 2.3 - 16/10/2008 AT 11:30 GMT.
    USE GALAHAD_IR_double                          ! double precision version
    USE GALAHAD_SMT_double
    USE GALAHAD_SILS_double
    IMPLICIT NONE
    INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )       ! set precision
    TYPE ( SMT_type ) :: A
    TYPE ( SILS_control ) :: CNTL
    TYPE ( SILS_ainfo ) :: AINFO
    TYPE ( SILS_finfo ) :: FINFO
    TYPE ( SILS_sinfo ) :: SINFO
    TYPE ( SILS_factors) :: FACTORS
    TYPE ( IR_data_type ) :: data
    TYPE ( IR_control_type ) :: control
    TYPE ( IR_inform_type ) :: inform
    INTEGER :: i
    REAL ( KIND = wp ), ALLOCATABLE, DIMENSION ( : ) :: B, X
    READ( 5, * ) A%n,  A%ne                      ! read dimensions from file
    ALLOCATE( B( A%n ), X( A%n ) )               ! Allocate space for A, b and x
    ALLOCATE( A%VAL( A%ne ), A%ROW( A%ne ), A%COL( A%ne ) )
    READ( 5, * ) ( A%ROW( i ), A%COL( i ), A%VAL( i ), i = 1, A%ne )
    READ( 5, * ) B                              ! read A and b from file
    CALL SILS_INITIALIZE( FACTORS, CNTL )        ! initialize SILS structures
    CALL SILS_ANALYSE( A, FACTORS, CNTL, AINFO )  ! order the rows of A
    IF ( AINFO%FLAG < 0 ) THEN                    ! check for errors
     WRITE( 6,'( A, I0 )' ) ' Failure of SILS_ANALYSE with flag = ', AINFO%FLAG
     STOP
    END IF
    CALL SILS_FACTORIZE( A, FACTORS, CNTL, FINFO ) ! factorize A
    IF( FINFO%FLAG < 0 ) THEN                     ! check for errors
     WRITE( 6,'( A, I0 )' ) ' Failure of SILS_FACTORIZE with flag = ', FINFO%FLAG
     STOP
    END IF
    CALL IR_initialize( data, control )          ! initialize IR structures
    control%itref_max = 2                        ! perform 2 iterations
    control%acceptable_residual_relative = 0.1 * EPSILON( 1.0D0 ) ! high accuracy
    X = B
    CALL IR_SOLVE( A, FACTORS, X, data, control, CNTL, inform ) ! find x
    IF ( inform%status == 0 ) THEN               ! check for errors
      WRITE( 6, '( A, /, ( 3F20.16 ) )' ) ' Solution after refinement is', X
    ELSE
     WRITE( 6,'( A, I2 )' ) ' Failure of IR_solve with status = ', inform%status
    END IF
    CALL IR_terminate( data, control, inform )    ! delete internal workspace
    CALL SILS_finalize( FACTORS, CNTL, i )
    END PROGRAM GALAHAD_IR_EXAMPLE
```

with the following data

```
5 7
1 1 2.0
1 2 3.0
2 3 4.0
2 5 6.0
3 3 1.0
3 4 5.0
```

```
5 5 1.0
8. 45. 31. 15. 17.
```

This produces the following output:

```
Solution after refinement is
 1.0000000000000000  2.0000000000000000  3.0000000000000000
 4.0000000000000000  5.0000000000000000
```