



Science and
Technology
Facilities Council



GALAHAD

UGO

USER DOCUMENTATION

GALAHAD Optimization Library version 5.0

1 SUMMARY

This package aims to find the **global minimizer of a univariate twice-continuously differentiable function $f(x)$ of a single variable over the finite interval $x^l \leq x \leq x^u$** . Function and derivative values may be provided either via a subroutine call, or by a return to the calling program. Second derivatives may be used to advantage if they are available.

ATTRIBUTES — Versions: GALAHAD_UGO_single, GALAHAD_UGO_double. **Uses:** GALAHAD_CLOCK, GALAHAD_SYMBOLS, GALAHAD_SPECFILE, GALAHAD_SPACE, GALAHAD_STRINGS, and GALAHAD_USERDATA. **Date:** May 2016. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory. **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_UGO_single
```

with the obvious substitution GALAHAD_UGO_double, GALAHAD_UGO_single_64 and GALAHAD_UGO_double_64 for the other variants.

If it is required to use more than one of the modules at the same time, the derived types UGO_time_type, UGO_control_type, UGO_inform_type, UGO_data_type and (Section 2.2) and the subroutines UGO_initialize, UGO_solve, UGO_terminate, (Section 2.3) and UGO_read_specfile (Section 2.7) 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 UGO_control_type is used to hold controlling data. Default values may be obtained by calling UGO_initialize (see Section 2.3.1), while components may also be changed by calling GALAHAD_UGO_read_spec (see Section 2.7.1). The components of UGO_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 UGO_solve and UGO_terminate is suppressed if error ≤ 0 . The default is error = 6.

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

`out` is a scalar variable of type `INTEGER(ip_)`, that holds the stream number for informational messages. Printing of informational messages in `UGO_solve` 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 = 1`, a single line of output will be produced for each iteration of the process. If `print_level ≥ 2`, this output will be increased to provide significant detail of each iteration. The default is `print_level = 0`.

`start_print` is a scalar variable of type `INTEGER(ip_)`, that specifies the first iteration for which printing will occur in `UGO_solve`. If `start_print` is negative, printing will occur from the outset. The default is `start_print = -1`.

`stop_print` is a scalar variable of type `INTEGER(ip_)`, that specifies the last iteration for which printing will occur in `UGO_solve`. If `stop_print` is negative, printing will occur once it has been started by `start_print`. The default is `stop_print = -1`.

`print_gap` is a scalar variable of type `INTEGER(ip_)`. Once printing has been started, output will occur once every `print_gap` iterations. If `print_gap` is no larger than 1, printing will be permitted on every iteration. The default is `print_gap = 1`.

`maxit` is a scalar variable of type `INTEGER(ip_)`, that holds the maximum number of iterations which will be allowed in `UGO_solve`. The default is `maxit = 1000`.

`initial_points` is a scalar variable of type `INTEGER(ip_)`, that gives the number of initial (uniformly-spaced) evaluation points; if `initial_points ; 2`, it will be reset to 2. The default is `initial_points = 2`.

`storage_increment` is a scalar variable of type `INTEGER(ip_)`, that specifies the increment size of additional workspace storage that will be allocated as needed; if `storage_increment ; 1000`, it will be reset to 1000. The default is `storage_increment = 1000`.

`buffer` is a scalar variable of type `INTEGER(ip_)`, that specifies the unit number for any out-of-core i/o when expanding workspace arrays. The default is `buffer = 70`.

`lipschitz_estimate_used` is a scalar variable of type `INTEGER(ip_)`, that specifies what sort of Lipschitz constant estimate will be used. Possible values are

1. the global constant is provided in `global_lipschitz_constant`.
2. the global constant is estimated.
3. local constants are estimated.

The default is `lipschitz_estimate_used = 3`.

`next_interval_selection` is a scalar variable of type `INTEGER(ip_)`, that specifies how the next interval for examination is chosen. Possible values are

1. traditional, based on the best overall predicted value
2. a local improvement algorithm is used (not currently implemented, and defaults to 1)

The default is `next_interval_selection = 1`.

`refine_with_newton` is a scalar variable of type `INTEGER(ip_)`, that specifies the maximum number of Newton steps that may be used in the vicinity of a global minimizer to try to improve the estimate. The default is `refine_with_newton = 5`.

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

`alive_unit` is a scalar variable of type `INTEGER(ip_)`. If `alive_unit > 0`, a temporary file named `alive_file` (see below) will be created on stream number `alive_unit` on initial entry to `GALAHAD_UGO_solve`, and execution of `GALAHAD_UGO_solve` will continue so long as this file continues to exist. Thus, a user may terminate execution simply by removing the temporary file from this unit. If `alive_unit ≤ 0`, no temporary file will be created, and execution cannot be terminated in this way. The default is `alive_unit = 40`.

`stop_length` is a scalar variable of type `REAL(rp_)`, that is used to assess convergence. The method will stop if all sub-intervals that could contain the global minimizer are smaller than `stop_length` in length. The default is `stop_length = 10-5`.

`small_g_for_newton` is a scalar variable of type `REAL(rp_)`, that is used to assess when to use a Newton correction. If the absolute value of the gradient (first derivative) of f is smaller than `small_g_for_newton`, the next evaluation point may be at a Newton estimate of a local minimizer. The default is `small_g_for_newton = 10-3`.

`small_g` is a scalar variable of type `REAL(rp_)`, that is used to assess when no Newton search is necessary. This will be the case if the absolute value of the gradient at the end of the interval search is smaller than `small_g`. The default is `small_g = 10-6`.

`obj_sufficient` is a scalar variable of type `REAL(rp_)`, that may be used to stop the iteration if a value of x is found for which the objective function is sufficiently small. Specifically, the search will stop if the objective function is smaller than `obj_sufficient`. The default is `obj_sufficient = - u^{-2}` , where u is `EPSILON(1.0)` (`EPSILON(1.0D0)` in `GALAHAD_UGO_double`).

`global_lipschitz_constant` is a scalar variable of type `REAL(rp_)`, that is used to specify the global Lipschitz constant for the gradient; a negative value indicates that the value is unknown. The default is `global_lipschitz_constant = -1.0`.

`reliability_parameter` is a scalar variable of type `REAL(rp_)`, that is used to boost insufficiently large estimates of the Lipschitz constant if necessary. If `reliability_parameter` is not positive, it will be reset to 1.2 when second derivatives are provided (see `second_derivatives_available` below, and 2.0 if they are not. The default is `reliability_parameter = -1.0`.

`lipschitz_lower_bound` is a scalar variable of type `REAL(rp_)`, that provides a lower bound on the Lipschitz constant for the gradient. This must be non-negative (and not zero unless the function is constant). The default is `lipschitz_lower_bound = 10-8`.

`cpu_time_limit` is a scalar variable of type `REAL(rp_)`, that is used to specify the maximum permitted CPU time. Any negative value indicates no limit will be imposed. The default is `cpu_time_limit = - 1.0`.

`clock_time_limit` is a scalar variable of type `REAL(rp_)`, that is used to specify the maximum permitted elapsed system clock time. Any negative value indicates no limit will be imposed. The default is `clock_time_limit = - 1.0`.

`second_derivatives_available` is a scalar variable of type default `LOGICAL`, that must be set `.TRUE.` when second derivatives of $f(x)$ are available, and `.FALSE.` otherwise. The package is generally more effective if second derivatives are available. The default is `second_derivatives_available = .TRUE..`

`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..`

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

`alive_file` is a scalar variable of type default CHARACTER and length 30, that gives the name of the temporary file whose removal from stream number `alive_unit` terminates execution of `GALAHAD_UGO_solve`. The default is `alive_unit = ALIVE.d`.

`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 = ""`.

2.2.2 The derived data type for holding timing information

The derived data type `UGO_time_type` is used to hold elapsed CPU and system clock times for the calculation. The components of `UGO_time_type` are:

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

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

2.2.3 The derived data type for holding informational parameters

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

`status` is a scalar variable of type `INTEGER(ip_)`, that gives the exit status of the algorithm. See Sections 2.5 and 2.6 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`.

`iter` is a scalar variable of type `INTEGER(ip_)`, that holds the number of iterations performed.

`f_eval` is a scalar variable of type `INTEGER(ip_)`, that gives the total number of evaluations of the objective function.

`g_eval` is a scalar variable of type `INTEGER(ip_)`, that gives the total number of evaluations of the first derivative of the objective function.

`h_eval` is a scalar variable of type `INTEGER(ip_)`, that gives the total number of evaluations of the second derivative of the objective function.

`dx_best` is a scalar variable of type `INTEGER(ip_)`, that gives the length of the largest remaining search interval.

`time` is a scalar variable of type `UGO_time_type` whose components are used to hold elapsed CPU and system clock times for the various parts of the calculation (see Section 2.2.2).

2.2.4 The derived data type for holding problem data

The derived data type `UGO_data_type` is used to hold all the data for a particular problem, or sequences of problems with the same structure, between calls of UGO procedures. This data should be preserved, untouched (except as directed on return from `GALAHAD_UGO_solve` with positive values of `inform%status`, see Section 2.5), from the initial call to `UGO_initialize` to the final call to `UGO_terminate`.

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

2.2.5 The derived data type for holding user data

The derived data type `GALAHAD_userdata_type` is available from the package `GALAHAD_userdata` to allow the user to pass data to and from user-supplied subroutines for function and derivative calculations (see Section 2.4). Components of variables of type `GALAHAD_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_UGO_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_UGO_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 three procedures for user calls (see Section 2.7 for further features):

1. The subroutine `UGO_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 `UGO_solve` is called to solve the problem.
3. The subroutine `UGO_terminate` is provided to allow the user to automatically deallocate array components of the private data, allocated by `UGO_solve`, at the end of the solution process.

We use square brackets [] to indicate OPTIONAL arguments.

2.3.1 The initialization subroutine

Default values are provided as follows:

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

`data` is a scalar `INTENT(INOUT)` argument of type `UGO_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 `UGO_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 `UGO_initialize`.

`inform` is a scalar `INTENT(OUT)` argument of type `UGO_inform_type` (see Section 2.2.3). A successful call to `UGO_initialize` is indicated when the component status has the value 0. For other return values of status, see Section 2.6.

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

2.3.2 The minimization subroutine

The minimization algorithm is called as follows:

```
CALL UGO_solve( x_l, x_u, x, f, g, h, control, inform, data, &
               userdata[, eval_FGH] )
```

- x_l and x_u are scalar `INTENT (IN)` variables of type `REAL(rp_)` that must be set to the lower and upper interval bounds x^l and x^u , respectively.
- x is a scalar `INTENT (INOUT)` variable of type `REAL(rp_)` that holds the next value of x at which the user is required to evaluate the objective (and its derivatives) when `inform%status > 0`, or the value of the approximate global minimizer when `inform%status = 0`. x need not be set on initial (`inform%status = 1`) entry.
- f is a scalar `INTENT (INOUT)` variable of type `REAL(rp_)` that must be set by the user to hold the value of the objective $f(x)$ at x given in x whenever `UGO_solve` returns with `inform%status > 0`. If `inform%status = 0`, f will contain the value of the approximate global minimum $f(x)$ at the approximate minimizer x given in x . f need not be set on initial (`inform%status = 1`) entry.
- g is a scalar `INTENT (INOUT)` variable of type `REAL(rp_)` that must be set by the user to hold the value of the first derivative $f'(x)$ of $f(x)$ at x given in x whenever `UGO_solve` returns with `inform%status > 0`. If `inform%status = 0`, g will contain the value of the gradient of $f(x)$ at the approximate minimizer x given in x . g need not be set on initial (`inform%status = 1`) entry.
- h is a scalar `INTENT (INOUT)` variable of type `REAL(rp_)` that must be set by the user to hold the value of the second derivative $f''(x)$ of $f(x)$ at x given in x whenever `UGO_solve` returns with `inform%status = 4` and the user has set `control%second_derivatives_available = .TRUE.` If `inform%status = 0`, h will contain the value of the second derivative of $f(x)$ at the approximate minimizer x given in x when second derivatives are available. h need not be set on initial (`inform%status = 1`) entry.
- `control` is a scalar `INTENT (IN)` argument of type `UGO_control_type` (see Section 2.2.1). Default values may be assigned by calling `UGO_initialize` prior to the first call to `UGO_solve`.
- `inform` is a scalar `INTENT (INOUT)` argument of type `UGO_inform_type` (see Section 2.2.3). On initial entry, the component `status` must be set to the value 1. Other entries need not be set. A successful call to `UGO_solve` is indicated when the component `status` has the value 0. For other return values of `status`, see Sections 2.5 and 2.6.
- `data` is a scalar `INTENT (INOUT)` argument of type `UGO_data_type` (see Section 2.2.4). It is used to hold data about the problem being solved. With the possible exceptions of the components `eval_status` and `U` (see Section 2.5), it must not have been altered **by the user** since the last call to `UGO_initialize`.
- `userdata` is a scalar `INTENT (INOUT)` argument of type `GALAHAD_userdata_type` whose components may be used to communicate user-supplied data to and from the `OPTIONAL` subroutine `eval_FGH` (see Section 2.2.5).
- `eval_FGH` is an `OPTIONAL` user-supplied subroutine whose purpose is to evaluate the value of the objective function $f(x)$ and its derivatives at a given vector x . See Section 2.4.1 for details. If `eval_FGH` is present, it must be declared `EXTERNAL` in the calling program. If `eval_FGH` is absent, `GALAHAD_UGO_solve` will use reverse communication to obtain objective function and derivative values (see Section 2.5).

2.3.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

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

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

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

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

`inform` is a scalar `INTENT(OUT)` argument of type `UGO_inform_type` exactly as for `UGO_solve`. Only the component `status` will be set on exit, and a successful call to `UGO_terminate` is indicated when this component `status` has the value 0. For other return values of `status`, see Section 2.6.

2.4 Function and derivative values

2.4.1 The objective function value via internal evaluation

If the argument `eval_FGH` is present when calling `GALAHAD_UGO_solve`, the user is expected to provide a subroutine of that name to evaluate the value of the objective function $f(x)$ and its first and (possibly) second derivatives, $f'(x)$ and $f''(x)$.

The routine must be specified as

```
SUBROUTINE eval_FGH( status, x, userdata, f, g, h )
```

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 objective function and its derivatives, and to a non-zero value if the evaluation has not been possible.

`x` is a scalar `INTENT(IN)` array argument of type `REAL(rp_)` whose components contain the value x .

`userdata` is a scalar `INTENT(INOUT)` argument of type `GALAHAD_userdata_type` whose components may be used to communicate user-supplied data to and from the subroutine (see Section 2.2.5).

`f` is a scalar `INTENT(OUT)` argument of type `REAL(rp_)`, that should be set to the value of the objective function $f(x)$ evaluated at the vector x input in `x`.

`g` is a scalar `INTENT(OUT)` argument of type `REAL(rp_)`, that should be set to the value of the first derivative $f'(x)$ of the objective function $f(x)$ evaluated at the vector x input in `x`.

`h` is a scalar `INTENT(OUT)` argument of type `REAL(rp_)`, that should be set to the value of the second derivative $f''(x)$ of the objective function $f(x)$ evaluated at the vector x input in `x` if `control%second_derivatives_available` has been set to `.TRUE.` It need not be set otherwise.

2.5 Reverse Communication Information

A positive value of `inform%status` on exit from `UGO_solve` indicates that `GALAHAD_UGO_solve` is seeking further information—this will happen if the user has chosen not to evaluate function or derivative values internally (see Section 2.4). The user should compute the required information and re-enter `GALAHAD_UGO_solve` with `inform%status` and all other arguments (except those specifically mentioned below) unchanged.

Possible values of `inform%status` and the information required are

3. The user should compute the objective function value $f(x)$ and its first derivative $f'(x)$ at the point x indicated in `x`. The required values should be set in `f` and `g`, respectively, and `data%eval_status` should be set to 0. If the user is unable to evaluate $f(x)$ or its first derivative—for instance, if the function is undefined at x —the user need not set `f` or `g`, but should then set `data%eval_status` to a non-zero value. This value can only occur when `control%second_derivatives_available = .FALSE.`

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

4. The user should compute the objective function value $f(x)$ and its first two derivatives $f'(x)$ and $f''(x)$ at the point x indicated in `x`. The required values should be set in `f`, `g` and `h`, respectively, and `data%eval_status` should be set to 0. If the user is unable to evaluate $f(x)$ or either of its derivatives—for instance, if the function is undefined at x —the user need not set `f`, `g` or `h`, but should then set `data%eval_status` to a non-zero value. This value can only occur when `control%second_derivatives_available = .TRUE.`

Other values are reserved for future developments.

2.6 Warning and error messages

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

- 1. A workspace 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 workspace 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.
- 4. The bound constraints are inconsistent.
- 7. The objective function appears to be unbounded from below on the feasible set.
- 18. Too many iterations have been performed. This may happen if `control%maxit` is too small, but may also be symptomatic of a badly scaled problem.
- 19. The elapsed CPU or system clock time limit has been reached. This may happen if either `control%cpu_time_limit` or `control%clock_time_limit` is too small, but may also be symptomatic of a badly scaled problem.
- 82. The user has forced termination of `GALAHAD-UGO_solve` by removing the file named `control%alive_file` from unit `control%alive_unit`.

2.7 Further features

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

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

```
( .. lines ignored by UGO_read_specfile .. )
BEGIN UGO
    keyword    value
```

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.


```

      .....      .....
      keyword      value
END
( .. lines ignored by UGO_read_specfile .. )

```

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

and

```
END UGO SPECIFICATION
```

are acceptable. Furthermore, between the “BEGIN UGO” 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 UGO_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 UGO_read_specfile.

2.7.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL UGO_read_specfile( control, device )
```

`control` is a scalar `INTENT(INOUT)` argument of type `UGO_control_type` (see Section 2.2.1). Default values should have already been set, perhaps by calling `UGO_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 on the following page.

`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`.

2.8 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`, a single line of output will be produced for each iteration of the process. This will include the value of the current iterate x and its corresponding objective function and gradient values, as well as the current best x and objective values. If `control%print_level = 2`, a final summary of the intervals considered is given, while if `control%print_level > 2`, additional debugging information will be given.

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

| command | component of control | value type |
|--------------------------------|----------------------------------|------------|
| error-printout-device | %error | integer |
| printout-device | %out | integer |
| print-level | %print_level | integer |
| start-print | %start_print | integer |
| stop-print | %stop_print | integer |
| iterations-between-printing | %print_gap | integer |
| maximum-number-of-iterations | %maxit | integer |
| number-of-initial-points | initial_points | integer |
| block-of-storage-allocated | storage_increment | integer |
| out-of-core-buffer | buffer | integer |
| lipschitz-estimate-used | lipschitz_estimate_used | integer |
| next-interval-selection-method | next_interval_selection | integer |
| refine-with-newton-iterations | refine_with_newton | integer |
| alive-device | %alive_unit | integer |
| stop_length | maximum-interval-length-required | real |
| small_g_for_newton | try-newton-tolerance | real |
| small_g | sufficient-gradient-tolerance | real |
| obj_sufficient | sufficient-objective-value | real |
| global-lipschitz-constant | global_lipschitz_constant | real |
| reliability_parameter | lipschitz-reliability-parameter | real |
| lipschitz_lower_bound | lipschitz-lower-bound | real |
| maximum-cpu-time-limit | %cpu_time_limit | real |
| maximum-clock-time-limit | %clock_time_limit | real |
| space-critical | %space_critical | logical |
| deallocate-error-fatal | %deallocate_error_fatal | logical |
| alive-filename | %alive_file | character |

Table 2.1: Specfile commands and associated components of control.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: UGO_solve calls the GALAHAD packages GALAHAD_CLOCK, GALAHAD_SYMBOLS, GALAHAD_SPECFILE, GALAHAD_SPACE, GALAHAD_STRINGS, and GALAHAD_USERDATA.

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

The algorithm starts by splitting the interval $[x^l, x^u]$ into a specified number of subintervals $[x_i, x_{i+1}]$ of equal length, and evaluating f and its derivatives at each x_i . A surrogate (approximating) lower bound function is constructed

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

on each subinterval using the function and derivative values at each end, and an estimate of the first- and second-derivative Lipschitz constant. This surrogate is minimized, the true objective evaluated at the best predicted point, and the corresponding interval split again at this point. Any interval whose surrogate lower bound value exceeds an evaluated actual value is discarded. The method continues until only one interval of a maximum permitted width remains.

References:

Many ingredients in the algorithm are based on the paper

Daniela Lera and Yaroslav D. Sergeyev, "Acceleration of univariate global optimization algorithms working with Lipschitz functions and Lipschitz first derivatives" SIAM J. Optimization Vol. 23, No. 1, pp. 508–529 (2013) but adapted to use second derivatives.

5 EXAMPLES OF USE

Suppose we wish to minimize the parametric objective function $f(\mathbf{x}) = x^2 \sin(ax)$ when the parameter a takes the value 10, and x is required to satisfy the bounds $-1 \leq x \leq 2$. We may use the following code:

```
PROGRAM GALAHAD_UGO_EXAMPLE ! GALAHAD 4.0 - 2022-03-07 AT 10:05 GMT
USE GALAHAD_UGO_double      ! double precision version
USE GALAHAD_USERDATA_double
IMPLICIT NONE
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
TYPE ( UGO_control_type ) :: control
TYPE ( UGO_inform_type ) :: inform
TYPE ( UGO_data_type ) :: data
TYPE ( GALAHAD_userdata_type ) :: userdata
EXTERNAL :: FGH
REAL ( KIND = wp ) :: x_l, x_u, x, f, g, h
REAL ( KIND = wp ), PARAMETER :: a = 10.0_wp
x_l = - 1.0_wp; x_u = 2.0_wp ! bounds on x
ALLOCATE( userdata%real( 1 ) ) ! Allocate space for parameter
userdata%real( 1 ) = a ! Record parameter, a
CALL UGO_initialize( data, control, inform )
inform%status = 1 ! set for initial entry
CALL UGO_solve( x_l, x_u, x, f, g, h, control, inform, data, userdata, &
eval_FGH = FGH ) ! Solve problem
IF ( inform%status == 0 ) THEN ! Successful return
WRITE( 6, "( ' UGO: ', I0, ' evaluations', /, &
& ' Optimal solution =', ES14.6, /, &
& ' Optimal objective value and gradient =', 2ES14.6 )" ) &
inform%iter, x, f, g
ELSE ! Error returns
WRITE( 6, "( ' UGO_solve exit status = ', I6 ) " ) inform%status
END IF
CALL UGO_terminate( data, control, inform ) ! delete internal workspace
END PROGRAM GALAHAD_UGO_EXAMPLE

SUBROUTINE FGH( status, x, userdata, f, g, h )
USE GALAHAD_USERDATA_double
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
INTEGER, INTENT( OUT ) :: status
REAL ( KIND = wp ), INTENT( IN ) :: x
```

All use is subject to the conditions of a BSD-3-Clause License.

See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.

```

REAL ( KIND = wp ), INTENT( OUT ) :: f, g, h
TYPE ( GALAHAD_userdata_type ), INTENT( INOUT ) :: userdata
REAL ( KIND = wp ) :: a, ax, sax, cax
a = userdata%real( 1 )
ax = a * x
sax = SIN( ax )
cax = COS( ax )
f = x * x * cax
g = - ax * x * sax + 2.0_wp * x * cax
h = - a * a * x * x * cax - 4.0_wp * ax * sax + 2.0_wp * cax
status = 0
RETURN
END SUBROUTINE FGH

```

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

```

UGO: 20 evaluations
Optimal solution = 1.583361E+00
Optimal objective value and gradient = -2.487269E+00 1.558900E-10

```

The same problem may be solved by returning to the user to provide the desired function and derivative values as follows:

```

PROGRAM GALAHAD_UGO_EXAMPLE2 ! GALAHAD 4.0 - 2022-03-07 AT 10:15 GMT
USE GALAHAD_UGO_double ! double precision version
USE GALAHAD_USERDATA_double
IMPLICIT NONE
INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
TYPE ( UGO_control_type ) :: control
TYPE ( UGO_inform_type ) :: inform
TYPE ( UGO_data_type ) :: data
TYPE ( GALAHAD_userdata_type ) :: userdata
REAL ( KIND = wp ) :: x_l, x_u, x, f, g, h
REAL ( KIND = wp ), PARAMETER :: a = 10.0_wp
REAL ( KIND = wp ) :: ax, sax, cax
x_l = - 1.0_wp; x_u = 2.0_wp ! bounds on x
ALLOCATE( userdata%real( 1 ) ) ! Allocate space for parameter
userdata%real( 1 ) = a ! Record parameter, a
CALL UGO_initialize( data, control, inform )
inform%status = 1 ! set for initial entry
DO ! Solve problem using reverse communication
CALL UGO_solve( x_l, x_u, x, f, g, h, control, inform, data, userdata )
SELECT CASE ( inform%status )
CASE ( 0 ) ! Successful return
WRITE( 6, "( ' UGO: ', I0, ' evaluations', /, &
& ' Optimal solution =', ES14.6, /, &
& ' Optimal objective value and gradient =', 2ES14.6 )" ) &
inform%iter, x, f, g
EXIT
CASE ( 4 ) ! obtain function/derivative values
ax = a * x
sax = SIN( ax )
cax = COS( ax )
f = x * x * cax
g = - ax * x * sax + 2.0_wp * x * cax
h = - a * a * x * x * cax - 4.0_wp * ax * sax + 2.0_wp * cax

```

All use is subject to the conditions of a BSD-3-Clause License.
See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.



```
data%eval_status = 0
CASE DEFAULT                                ! Error returns
  WRITE( 6, "( ' UGO_solve exit status = ', I6 ) " ) inform%status
  EXIT
END SELECT
END DO
CALL UGO_terminate( data, control, inform ) ! delete internal workspace
END PROGRAM GALAHAD_UGO_EXAMPLE2
```

This produces the same output.