C interfaces to GALAHAD UGO

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory

Sun Mar 20 2022

Chapter 1

GALAHAD C package ugo

1.1 Introduction

1.1.1 Purpose

The ugo 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.

1.1.2 Authors

N. I. M. Gould, STFC-Rutherford Appleton Laboratory, England.

C interface, additionally J. Fowkes, STFC-Rutherford Appleton Laboratory.

1.1.3 Originally released

July 2016, C interface August 2021.

1.1.4 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 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.

1.1.5 References

Many ingredients in the algorithm are based on the paper

D. Lera and Ya. D. Sergeyev (2013), `‘Acceleration of univariate global optimization algorithms working with Lipschitz functions and Lipschitz first derivatives'' SIAM J. Optimization Vol. 23, No. 1, pp. 508–529,

but adapted to use second derivatives.

1.2 Call order

To solve a given problem, functions from the ugo package must be called in the following order:

- ugo_initialize - provide default control parameters and set up initial data structures
- ugo_read_specfile (optional) - override control values by reading replacement values from a file
- ugo_import - set up problem data structures and fixed values
- ugo_reset_control (optional) - possibly change control parameters if a sequence of problems are being solved
- solve the problem by calling one of
  - ugo_solve_direct - solve using function calls to evaluate function and derivative values, or
  - ugo_solve_reverse - solve returning to the calling program to obtain function and derivative values
- ugo_information (optional) - recover information about the solution and solution process
- ugo_terminate - deallocate data structures

See Section 4.1 for examples of use.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 ugo.h File Reference

```
#include <stdbool.h>
#include "galahad_precision.h"
```

Data Structures

- struct ugo_control_type
- struct ugo_time_type
- struct ugo_inform_type

Functions

- void ugo_initialize (void **data, struct ugo_control_type *control, int *status)
- void ugo_read_specfile (struct ugo_control_type *control, const char specfile[ ])
- void ugo_import (struct ugo_control_type *control, void **data, int *status, const real_wp_ *x_l, const real←_wp_ *x_u)
- void ugo_reset_control (struct ugo_control_type *control, void **data, int *status)
- void ugo_solve_direct (void **data, void *userdata, int *status, real_wp_ *x, real_wp_ *f, real_wp_ *g, real←_wp_ *h, int(*eval_fgh)(real_wp_, real_wp_ *, real_wp_ *, real_wp_ *, const void *))
- void ugo_solve_reverse (void **data, int *status, int *eval_status, real_wp_ *x, real_wp_ *f, real_wp_ *g, real_wp_ *h)
- void ugo_information (void **data, struct ugo_inform_type *inform, int *status)
- void ugo_terminate (void **data, struct ugo_control_type *control, struct ugo_inform_type *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct ugo_control_type

Examples

ugos.c, and ugot.c.

Data Fields

int	error	error and warning diagnostics occur on stream error
int	out	general output occurs on stream out
int	print_level	the level of output required. Possible values are: • $\leq$ 0 no output, • 1 a one-line summary for every improvement • 2 a summary of each iteration • $\geq$ 3 increasingly verbose (debugging) output
int	start_print	any printing will start on this iteration
int	stop_print	any printing will stop on this iteration
int	print_gap	the number of iterations between printing
int	maxit	the maximum number of iterations allowed
int	initial_points	the number of initial (uniformly-spaced) evaluation points ($<$2 reset to 2)
int	storage_increment	incremenets of storage allocated (less that 1000 will be reset to 1000)
int	buffer	unit for any out-of-core writing when expanding arrays
int	lipschitz_estimate_used	what sort of Lipschitz constant estimate will be used: • 1 = global contant provided • 2 = global contant estimated • 3 = local costants estimated
int	next_interval_selection	how is the next interval for examination chosen: • 1 = traditional • 2 = local_improvement
int	refine_with_newton	try refine_with_newton Newton steps from the vacinity of the global minimizer to try to improve the estimate
int	alive_unit	removal of the file alive_file from unit alive_unit terminates execution
char	alive_file[31]	see alive_unit
real_wp_	stop_length	overall convergence tolerances. The iteration will terminate when the step is less than .stop_length
real_wp_	small_g_for_newton	if the absolute value of the gradient is smaller than small_g_for_newton, the next evaluation point may be at a Newton estimate of a local minimizer
real_wp_	small_g	if the absolute value of the gradient at the end of the interval search is smaller than small_g, no Newton serach is necessary
real_wp_	obj_sufficient	stop if the objective function is smaller than a specified value
real_wp_	global_lipschitz_constant	the global Lipschitz constant for the gradient (-ve means unknown)
real_wp_	reliability_parameter	the reliability parameter that is used to boost insufficiently large estimates of the Lipschitz constant (-ve means that default values will be chosen depending on whether second derivatives are provided or not)
real_wp_	lipschitz_lower_bound	a lower bound on the Lipscitz constant for the gradient (not zero unless the function is constant)

Data Fields

real_wp_	cpu_time_limit	the maximum CPU time allowed (-ve means infinite)
real_wp_	clock_time_limit	the maximum elapsed clock time allowed (-ve means infinite)
bool	second_derivative_available	if .second_derivative_available is true, the user must provide them when requested. The package is generally more effective if second derivatives are available.
bool	space_critical	if .space_critical is true, every effort will be made to use as little space as possible. This may result in longer computation time
bool	deallocate_error_fatal	if .deallocate_error_fatal is true, any array/pointer deallocation error will terminate execution. Otherwise, computation will continue
char	prefix[31]	all output lines will be prefixed by .prefix(2:LEN(TRIM(.prefix))-1) where .prefix contains the required string enclosed in quotes, e.g. "string" or 'string'

3.1.1.2 struct ugo_time_type

Data Fields

real_sp_	total	the total CPU time spent in the package
real_wp_	clock_total	the total clock time spent in the package

3.1.1.3 struct ugo_inform_type

Examples

ugos.c, and ugot.c.

Data Fields

int	status	return status. See UGO_solve for details
int	eval_status	evaluation status for reverse communication interface
int	alloc_status	the status of the last attempted allocation/deallocation
char	bad_alloc[81]	the name of the array for which an allocation/deallocation error ocurred
int	iter	the total number of iterations performed
int	f_eval	the total number of evaluations of the objection function
int	g_eval	the total number of evaluations of the gradient of the objection function
int	h_eval	the total number of evaluations of the Hessian of the objection function
struct ugo_time_type	time	timings (see above)

3.1.2 Function Documentation

3.1.2.1 ugo_initialize()

```
void ugo_initialize (
            void ** data,
```

```
            struct ugo_control_type * control,
            int * status )
```

Set default control values and initialize private data

Parameters

in,out	*data*	holds private internal data
out	*control*	is a struct containing control information (see ugo_control_type)
out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): • 0. The import was succesful.

Examples

ugos.c, and ugot.c.

3.1.2.2 ugo_read_specfile()

```
void ugo_read_specfile (
            struct ugo_control_type * control,
            const char specfile[] )
```

Read the content of a specification file, and assign values associated with given keywords to the corresponding control parameters. By default, the spcification file will be named RUNUGO.SPC and lie in the current directory. Refer to Table 2.1 in the fortran documentation provided in $GALAHAD/doc/ugo.pdf for a list of keywords that may be set.

Parameters

in,out	*control*	is a struct containing control information (see ugo_control_type)
in	*specfile*	is a character string containing the name of the specification file

Examples

ugot.c.

3.1.2.3 ugo_import()

```
void ugo_import (
            struct ugo_control_type * control,
            void ** data,
            int * status,
            const real_wp_ * x_l,
            const real_wp_ * x_u )
```

Import problem data into internal storage prior to solution.

Parameters

in	*control*	is a struct whose members provide control paramters for the remaining prcedures (see ugo_control_type)
in,out	*data*	holds private internal data
in,out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are: • 1. The import was succesful, and the package is ready for the solve phase • -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.
in	*x_l*	is a scalar variable of type double, that holds the value x^l of the lower bound on the optimization variable x.
in	*x_u*	is a scalar variable of type double, that holds the value x^u of the upper bound on the optimization variable x.

Examples

ugos.c, and ugot.c.

3.1.2.4 ugo_reset_control()

```
void ugo_reset_control (
            struct ugo_control_type * control,
            void ** data,
            int * status )
```

Reset control parameters after import if required.

Parameters

in	*control*	is a struct whose members provide control paramters for the remaining prcedures (see ugo_control_type)
in,out	*data*	holds private internal data
in,out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are: • 1. The import was succesful, and the package is ready for the solve phase

3.1.2.5 ugo_solve_direct()

```
void ugo_solve_direct (
            void ** data,
            void * userdata,
            int * status,
            real_wp_ * x,
            real_wp_ * f,
            real_wp_ * g,
            real_wp_ * h,
            int(*)(real_wp_, real_wp_ *, real_wp_ *, real_wp_ *, const void *) eval_fgh )
```

Find an approximation to the global minimizer of a given univariate function with a Lipschitz gradient in an interval.

This version is for the case where all function/derivative information is available by function calls.

Parameters

in,out	*data*	holds private internal data
in	*userdata*	is a structure that allows data to be passed into the function and derivative evaluation programs (see below).
in,out	*status*	is a scalar variable of type int, that gives the entry and exit status from the package. On initial entry, status must be set to 1. Possible exit are: • 0. The run was succesful • -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. • -7. The objective function appears to be unbounded from below • -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 CPU time limit has been reached. This may happen if control.cpu_time_limit is too small, but may also be symptomatic of a badly scaled problem. • -40. The user has forced termination of solver by removing the file named control.alive_file from unit unit control.alive_unit.
out	*x*	is a scalar variable of type double, that holds the value of the approximate global minimizer x after a successful (status = 0) call.
out	*f*	is a scalar variable of type double, that holds the the value of the objective function $f(x)$ at the approximate global minimizer x after a successful (status = 0) call.
out	*g*	is a scalar variable of type double, that holds the the value of the gradient of the objective function $f'(x)$ at the approximate global minimizer x after a successful (status = 0) call.
out	*h*	is a scalar variable of type double, that holds the the value of the second derivative of the objective function $f''(x)$ at the approximate global minimizer x after a successful (status = 0) call.

Parameters

	eval_fgh	is a user-provided function that must have the following signature:
		`int eval_fgh( double x, double *f,`
		` double *g,`
		` double *h,`
		` const void *userdata)`
		The value of the objective function $f(x)$ and its first derivative $f'(x)$ evaluated at x= x must be assigned to f and g respectively, and the function return value set to 0. In addition, if control.second_derivatives_available has been set to true, when calling ugo_import, the user must also assign the value of the second derivative $f''(x)$ in h; it need not be assigned otherwise. If the evaluation is impossible at x, return should be set to a nonzero value.

Examples

ugos.c.

3.1.2.6 ugo_solve_reverse()

```
void ugo_solve_reverse (
            void ** data,
            int * status,
            int * eval_status,
            real_wp_ * x,
            real_wp_ * f,
            real_wp_ * g,
            real_wp_ * h )
```

Find an approximation to the global minimizer of a given univariate function with a Lipschitz gradient in an interval.

This version is for the case where function/derivative information is only available by returning to the calling procedure.

Parameters

in,out	*data*	holds private internal data

Parameters

`in,out`	*status*	is a scalar variable of type int, that gives the entry and exit status from the package. On initial entry, status must be set to 1. Possible exit are: • 0. The run was succesful • -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. • -7. The objective function appears to be unbounded from below • -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 CPU time limit has been reached. This may happen if control.cpu_time_limit is too small, but may also be symptomatic of a badly scaled problem. • -40. The user has forced termination of solver by removing the file named control.alive_file from unit unit control.alive_unit. • 3. The user should compute the objective function value $f(x)$ and its first derivative $f'(x)$, and then re-enter the function. The required values should be set in f and g respectively, and eval_status (below) should be set to 0. If the user is unable to evaluate $f(x)$ or $f'(x)$ - for instance, if the function or its first derivative are undefined at x - the user need not set f or g, but should then set eval_status to a non-zero value. This value can only occur when control.second_derivatives_available = false. • 4. The user should compute the objective function value $f(x)$ and its first two derivatives $f'(x)$ and $f''(x)$ at x= x, and then re-enter the function. The required values should be set in f, g and h respectively, and eval_status (below) should be set to 0. If the user is unable to evaluate $f(x)$, $f'(x)$ or $f''(x)$ - for instance, if the function or its derivatives are undefined at x - the user need not set f, g or h, but should then set eval_status to a non-zero value. This value can only occur when control.second_derivatives_available = true.
`in,out`	*eval_status*	is a scalar variable of type int, that is used to indicate if objective function and its derivatives can be provided (see above).
`out`	*x*	is a scalar variable of type double, that holds the next value of x at which the user is required to evaluate the objective (and its derivatives) when status $>$ 0, or the value of the approximate global minimizer when status = 0
`in,out`	*f*	is a scalar variable of type double, that must be set by the user to hold the value of $f(x)$ if required by status $>$ 0 (see above), and will return the value of the approximate global minimum when status = 0
`in,out`	*g*	is a scalar variable of type double, that must be set by the user to hold the value of $f'(x)$ if required by status $>$ 0 (see above), and will return the value of the first derivative of f at the approximate global minimizer when status = 0

Parameters

in,out	*h*	is a scalar variable of type double, that must be set by the user to hold the value of $f''(x)$ if required by status $>$ 0 (see above), and will return the value of the second derivative of f at the approximate global minimizer when status = 0

Examples

ugot.c.

3.1.2.7 ugo_information()

```
void ugo_information (
            void ** data,
            struct ugo_inform_type * inform,
            int * status )
```

Provides output information

Parameters

in,out	*data*	holds private internal data
out	*inform*	is a struct containing output information (see ugo_inform_type)
out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): • 0. The values were recorded succesfully

Examples

ugos.c, and ugot.c.

3.1.2.8 ugo_terminate()

```
void ugo_terminate (
            void ** data,
            struct ugo_control_type * control,
            struct ugo_inform_type * inform )
```

Deallocate all internal private storage

Parameters

in,out	*data*	holds private internal data
out	*control*	is a struct containing control information (see ugo_control_type)
out	*inform*	is a struct containing output information (see ugo_inform_type)

Examples

ugos.c, and ugot.c.

Chapter 4

Example Documentation

4.1 ugos.c

This is an example of how to use the package to find an approximation to the global minimum of a given univariate function over an interval.

```
/* ugos.c */
/* Spec test for the UGO C interface */
#include <stdio.h>
#include <math.h>
#include "ugo.h"
struct userdata_type {
   double a;
};
// Evaluate test problem objective, first and second derivatives
int fgh(double x, double *f, double *g, double *h, const void *userdata){
   struct userdata_type *myuserdata = (struct userdata_type *) userdata;
   double a = myuserdata->a;
   *f = x * x * cos( a*x );
   *g = - a * x * x * sin( a*x ) + 2.0 * x * cos( a*x );
   *h = - a * a* x * x * cos( a*x ) - 4.0 * a * x * sin( a*x )
        + 2.0 * cos( a*x );
   return 0;
}
int main(void) {
    // Derived types
    void *data;
    struct ugo_control_type control;
    struct ugo_inform_type inform;
    // Initialize UGO
    ugo_initialize( &data, &control, &inform );
    // Set user-defined control options
    control.print_level = 1;
    control.maxit = 100;
    control.lipschitz_estimate_used = 3;
    // User data
    struct userdata_type userdata;
    userdata.a = 10.0;
    // Test problem bounds
    double x_l = -1.0;
    double x_u = 2.0;
    // Test problem objective, gradient, Hessian values
    double x, f, g, h;
    int status;
    // import problem data
    ugo_import( &control, &data, &status, &x_l, &x_u );
    // Set for initial entry
    status = 1;

    // Call UGO_solve
    ugo_solve_direct( &data, &userdata, &status, &x, &f, &g, &h, fgh );
    // Record solution information
    ugo_information( &data, &inform, &status );
    if(inform.status == 0){ // successful return
        printf("UGO successful solve\n");
        printf("iter: %d \n", inform.iter);
```

```
        printf("x: %f \n", x);
        printf("f: %f \n", f);
        printf("g: %f \n", g);
        printf("h: %f \n", h);
        printf("f_eval: %d \n", inform.f_eval);
        printf("time: %f \n", inform.time.clock_total);
        printf("status: %d \n", inform.status);
    }else{ // error returns
        printf("UGO error in solve\n");
        printf("status: %d \n", inform.status);
    }
    // Delete internal workspace
    ugo_terminate( &data, &control, &inform );
    return 0;
}
```

4.2 ugot.c

This is the same example, but now function and derivative information is found by reverse communication with the calling program.

```
/* ugo_test.c */
/* Simple code to test the UGO reverse communication C interface */
#include <stdio.h>
#include <math.h>
#include "ugo.h"
// Test problem objective
double objf(double x){
    double a = 10.0;
    return x * x * cos( a*x );
}
// Test problem first derivative
double gradf(double x){
    double a = 10.0;
    return - a * x * x * sin( a*x ) + 2.0 * x * cos( a*x );
}
// Test problem second derivative
double hessf(double x){
    double a = 10.0;
    return - a * a* x * x * cos( a*x ) - 4.0 * a * x * sin( a*x )
            + 2.0 * cos( a*x );
}
int main(void) {
    // Derived types
    void *data;
    struct ugo_control_type control;
    struct ugo_inform_type inform;
    // Initialize UGO
    int status, eval_status;
    ugo_initialize( &data, &control, &status );
    // Set user-defined control options
    //control.print_level = 1;
    //control.maxit = 100;
    //control.lipschitz_estimate_used = 3;
    // Read options from specfile
    char specfile[] = "UGO.SPC";
    ugo_read_specfile(&control, specfile);
    // Test problem bounds
    double x_l = -1.0;
    double x_u = 2.0;
    // Test problem objective, gradient, Hessian values
    double x, f, g, h;
    // import problem data
    ugo_import( &control, &data, &status, &x_l, &x_u );
    // Set for initial entry
    status = 1;
    // Solve the problem: min f(x), x_l <= x <= x_u
    while(true){
        // Call UGO_solve
        ugo_solve_reverse(&data, &status, &eval_status, &x, &f, &g, &h );
        // Evaluate f(x) and its derivatives as required
        if(status >= 2){ // need objective
            f = objf(x);
            if(status >= 3){ // need first derivative
                g = gradf(x);
                if(status >= 4){ // need second derivative
                    h = hessf(x);
                }
            }
        } else { // the solution has been found (or an error has occured)
```

```
            break;
        }
    }
    // Record solution information
    ugo_information( &data, &inform, &status );
    if(inform.status == 0){
        printf("%i evaluations. Optimal objective value = %5.2f"
          " status = %1i\n", inform.f_eval, f, inform.status);
    }else{
        printf("BGO_solve exit status = %1i\n", inform.status);
    }
    // Print solution details
    // printf("iter: %d \n", inform.iter);
    // printf("x: %f \n", x);
    // printf("f: %f \n", f);
    // printf("g: %f \n", g);
    // printf("h: %f \n", h);
    // printf("f_eval: %d \n", inform.f_eval);
    // printf("time: %f \n", inform.time.clock_total);
    // printf("status: %d \n", inform.status);
    // Delete internal workspace
    ugo_terminate( &data, &control, &inform );
    return 0;
}
```

Index