GKLS-Generator of Classes of Test Functions with Known Local and Global Minima for Global Optimization: USER MANUAL

MARCO GAVIANO
Università di Cagliari, Italy
DMITRI E. KVASOV
Università di Roma "La Sapienza", Italy, and University of Nizhni Novgorod, Russia
DANIELA LERA
Università di Cagliari, Italy
and
YAROSLAV D. SERGEYEV*
Università della Calabria, Italy, and University of Nizhni Novgorod, Russia

---

Contents

---

*Corresponding author, e-mail: yaro@si.deis.unical.it
Authors' addresses: M. Gaviano and D. Lera, Dipartimento di Matematica, Università di Cagliari, Via Ospedale, 72, 09100 Cagliari – Italy; D.E. Kvasov and Ya.D. Sergeyev, DEIS, Università della Calabria, Via P.Bucci, Cubo 41C, 87036 Rende (CS) – Italy.
This work has been partially supported by the following projects: FIRB RBAU01JYPN, FIRB RBNE01WBBB, and RFBR 01-01-00587.

1. INTRODUCTION

This *User Manual* describes an implementation of the GKLS-generator. First, the general structure of the package is presented, then instructions for using the test classes generator are described. The *User Manual* is strongly connected to the paper Gaviano et al. [2003] (called hereafter *paper*). Particularly, any formulae referenced in the *User Manual* are to be found in the *paper*. In the *paper*, the user can find an example of a generated test function also.

2. STRUCTURE OF THE PACKAGE

The GKLS-generator package has been written in ANSI Standard C and successfully tested on Windows and UNIX platforms. Our implementation follows the procedure described in Section 3 of the *paper*.

The package implementing the generator includes the following files:

gkls.c – the main file;

gkls.h – the header file that users should include in their application projects in order to call subroutines from the file **gkls.c**;

rnd_gen.c – the file containing the uniform random number generator proposed in Knuth [1997; 2002];

rnd_gen.h – the header file for linkage to the file **rnd_gen.c**;

example.c – an example of the GKLS-generator usage;

Makefile – an example of a UNIX makefile provided to UNIX users for a simple compilation and linkage of separate files of the application project.

For implementation details the user can consult the C codes. Note that the random number generator in **rnd_gen.c** uses the logical-and operation '&' for efficiency, so it is not strictly portable unless the computer uses two's complement representation for integer. It does not limit portability of the package because almost all modern computers are based on two's complement arithmetic.

3. CALLING SEQUENCE FOR GENERATION AND USAGE OF THE TESTS CLASSES

Here we describe how to generate and use classes of non-differentiable (ND-type), differentiable (D-type), and twice continuously differentiable (D2-type) test functions. Again, we concentrate on the D-type functions. The operations for the remaining two types are analogous.

To utilize the GKLS-generator the user must perform the following steps:

Step 1. Input of the parameters defining a specific test class.

Step 2. Generating a specific test function of the defined test class.

Step 3. Evaluation of the generated test function and, if necessary, its partial derivatives.

Step 4. Memory deallocating.

Let us consider these steps in turn.

3.1 Input of the parameters defining a specific test class

This step is subdivided into: (1) defining the parameters of the test class, (2) defining the admissible region Ω, and (3) checking (if necessary).

3.1.1 *Defining the parameters of the test class.* The parameters to be defined by the user determine a specific class (of the ND-, D- or D2-type) of 100 test functions (a specific function is retrieved by its number). There are the following parameters:

GKLS_dim – (**unsigned int**) dimension N (from (1)) of test functions; $N \geq 2$ (since multidimensional problems are considered in (1)) and $N <$ NUM_RND in **rnd_gen.h**; this value is limited by the power of **unsigned int**-representation; default $N = 2$;

GKLS_num_minima – (**unsigned int**) number m (from (4)) of local minima including the paraboloid Z minimum (from (3)) and the global minimum; $m \geq 2$; the upper bound of this parameter is limited by the power of **unsigned int**-representation; default $m = 10$;

GKLS_global_value – (**double**) global minimum value f^* of $f(x)$; condition (16) must be satisfied; the default value is -1.0 (defined in the file **gkls.h** as a constant GKLS_GLOBAL_MIN_VALUE);

GKLS_global_dist – (**double**) distance r^* from the paraboloid vertex T in (3) to the global minimizer $x^* \in \Omega$ of $f(x)$; condition (17) must be satisfied; the default value is

$$\text{GKLS\_global\_dist} \stackrel{\text{def}}{=} \min_{1 \leq j \leq N} |b(j) - a(j)|/\,3,$$

where the vectors a and b determine the admissible region Ω in (2);

GKLS_global_radius – (**double**) radius ρ^* of the attraction region of the global minimizer $x^* \in \Omega$ of $f(x)$; condition (18) must be satisfied; the default value is

$$\text{GKLS\_global\_radius} \stackrel{\text{def}}{=} \min_{1 \leq j \leq N} |b(j) - a(j)|/\,6.$$

The user may call subroutine *GKLS_set_default*() to set the default values of these five variables.

3.1.2 *Defining the admissible region.* With N determined, the user must allocate dynamic arrays *GKLS_domain_left* and *GKLS_domain_right* to define the boundary of the hyperrectangle Ω. This is done by calling subroutine

int *GKLS_domain_alloc* ();

which has no parameters and returns the following error codes defined in **gkls.h**:

GKLS_OK – no errors;

GKLS_DIM_ERROR – the problem dimension is out of range; it must be greater than or equal to 2 and less than NUM_RND defined in **rnd_gen.h**;

GKLS_MEMORY_ERROR – there is not enough memory to allocate.

The same subroutine defines the admissible region Ω. The default value $\Omega = [-1, 1]^N$ is set by *GKLS_set_default*().

3.1.3 *Checking.* The following subroutine allows the user to check validity of the input parameters:

int *GKLS_parameters_check* ().

It has no parameters and returns the following error codes (see **gkls.h**):

GKLS_OK – no errors;

GKLS_DIM_ERROR – problem dimension error;

GKLS_NUM_MINIMA_ERROR – number of local minima error;

GKLS_BOUNDARY_ERROR – the admissible region boundary vectors are ill-defined;

GKLS_GLOBAL_MIN_VALUE_ERROR – the global minimum value is not less than the paraboloid (3) minimum value t defined in **gkls.h** as a constant GKLS_PARABOLOID_MIN;

GKLS_GLOBAL_DIST_ERROR – the parameter r^* does not satisfy (17);

GKLS_GLOBAL_RADIUS_ERROR – the parameter ρ^* does not satisfy (18).

3.2 Generating a specific test function of the defined test class

After a specific test class has been chosen (i.e., the input parameters have been determined) the user can generate a specific function that belongs to the chosen class of 100 test functions. This is done by calling subroutine

int *GKLS_arg_generate* (**unsigned int** *nf*);

where

nf – the number of a function from the test class (from 1 to 100).

This subroutine initializes the random number generator, checks the input parameters, allocates dynamic arrays, and generates a test function following the procedure of Section 3 of the *paper*. It returns an error code that can be the same as for subroutines *GKLS_parameters_check*() and *GKLS_domain_alloc*(), or additionally:

GKLS_FUNC_NUMBER_ERROR – the number of a test function to generate exceeds 100 or it is less than 1.

GKLS_arg_generate() generates the list of all local minima and the list of the global minima as parts of the structures *GKLS_minima* and *GKLS_glob*, respectively. The first structure gathers the following information about all local minima (including the paraboloid minimum and the global one): coordinates of local minimizers, local minima values, and attraction regions radii. The second structure contains information about the number of global minimizers and their indices in the set of local minimizers. It has the following fields:

num_global_minima – (**unsigned int**) total number of global minima;

gm_index – (**unsigned int ***) list of indices of generated minimizers, which are the global ones (elements 0 to ($num\_global\_minima - 1$) of the list) and the local ones (the remaining elements of the list).

The elements of the list *GKLS_glob.gm_index* are indices to a specific minimizer in the first structure *GKLS_minima* characterized by the following fields:

local_min – (**double ****) list of local minimizers coordinates;

f – (**double ***) list of local minima values;

rho – (**double ***) list of attraction regions radii;

peak – (**double ***) list of parameters γ_i values from (10);

w_rho – (**double ***) list of parameters w_i values from (23).

The fields of these structures can be useful if one needs to study properties of a specific generated test function more deeply.

3.3 Evaluation of a generated test function or its partial derivatives

While there exists a structure *GKLS_minima* of local minima, the user can evaluate a test function (or partial derivatives of D- and D2-type functions) that is determined by its number (a parameter to the subroutine *GKLS_arg_generate*()) within the chosen test class. If the user wishes to evaluate another function within the same class he should deallocate dynamic arrays (see the next subsection) and recall the generator *GKLS_arg_generate*() (passing it the corresponding function number) without resetting the input class parameters (see subsection 3.1). If the user wishes to change the test class properties he should reset also the input class parameters.

Evaluation of an ND-type function is done by calling subroutine
double *GKLS_ND_func* (x).
Evaluation of a D-type function is done by calling subroutine
double *GKLS_D_func* (x).
Evaluation of a D2-type function is done by calling subroutine
double *GKLS_D2_func* (x).
All these subroutines have only one input parameter

x – (**double ***) a point $x \in \mathbb{R}^N$ where the function must be evaluated.

All the subroutines return a test function value corresponding to the point x. They return the value GKLS_MAX_VALUE (defined in **gkls.h**) in two cases: (a) vector x does not belong to the admissible region Ω and (b) the user tries to call the subroutines without generating a test function.

The following subroutines are provided for calculating the partial derivatives of the test functions (see Appendix in the *paper*).

Evaluation of the first order partial derivative of the D-type test functions with respect to the variable x_j (see (A.1)–(A.2) in Appendix) is done by calling subroutine
double *GKLS_D_deriv* (j, x).
Evaluation of the first order partial derivative of the D2-type test functions with respect to the variable x_j (see (A.3)–(A.4) in Appendix) is done by calling subroutine
double *GKLS_D2_deriv1* (j, x).
Evaluation of the second order partial derivative of the D2-type test functions with respect to the variables x_j and x_k (see in Appendix the formulae (A.5)–(A.6) for the case $j \neq k$ and (A.7)–(A.8) for the case $j = k$) is done by calling subroutine
double *GKLS_D2_deriv2* (j, k, x).
Input parameters for these three subroutines are:

j, k – (**unsigned int**) indices of the variables (that must be in the range from 1 to *GKLS_dim*) with respect to which the partial derivative is evaluated;

x – (**double ***) a point $x \in \mathbb{R}^N$ where the derivative must be evaluated.

All subroutines return the value of a specific partial derivative corresponding to the point x and to the given direction. They return the value GKLS_MAX_VALUE (defined in **gkls.h**) in three cases: (a) index (j or k) of a variable is out of the range $[1, \mathit{GKLS\_dim}]$; (b) vector x does not belong to the admissible region Ω; (c) the user tries to call the subroutines without generating a test function.

Subroutines for calculating the gradients of the D- and D2-type test functions and for calculating the Hessian matrix of the D2-type test functions at a given feasible point are also provided. These are

int *GKLS_D_gradient* (x, g),

int *GKLS_D2_gradient* (x, g),

int *GKLS_D2_hessian* (x, h).

Here

x – (**double ***) a point $x \in \mathbb{R}^N$ where the gradient or Hessian matrix must be evaluated;

g – (**double ***) a pointer to the gradient vector calculated at x;

h – (**double ****) a pointer to the Hessian matrix calculated at x.

Note that before calling these subroutines the user must allocate dynamic memory for the gradient vector g or the Hessian matrix h and pass the pointers g or h as parameters of the subroutines.

These subroutines call the subroutines described above for calculating the partial derivatives and return an error code (**GKLS_DERIV_EVAL_ERROR** in the case of an error during evaluation of a particular component of the gradient or the Hessian matrix, or **GKLS_OK** if there are no errors).

3.4 Memory deallocating

When the user concludes his work with a test function he should deallocate dynamic arrays allocated by the generator. This is done by calling subroutine

void *GKLS_free* (**void**);

with no parameters.

When the user abandons the test class he should deallocate dynamic boundaries vectors *GKLS_domain_left* and *GKLS_domain_right* by calling subroutine

void *GKLS_domain_free* (**void**);

again with no parameters.

It should be finally highlighted that if the user, after deallocating memory, wishes to return to the same class, generation of the class with the same parameters produces the same 100 test functions.

An example of the generation and use of some of the test classes can be found in the file **example.c**.

REFERENCES

GAVIANO, M., KVASOV, D. E., LERA, D., AND SERGEYEV, Y. D. *2003. Generation of classes of test functions with known local minima.* ACM Trans. Math. Soft. V, *N, PP–PP.*

KNUTH, D. *1997.* The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, third ed. Addison-Wesley, Reading, Massachusetts.*

KNUTH, D. *2002. Home page at: http://sunburn.stanford.edu/~ knuth/.*