

**NAME**

CUTEST\_csetup\_threaded – CUTEst tool to set up the data structures for constrained minimization.

**SYNOPSIS**

CALL CUTEST\_csetup\_threaded( status, input, out, threads, IO\_BUFFERS, n, m, X, X\_l, X\_u, Y, C\_l, C\_u, EQUATN, LINEAR, e\_order, l\_order, v\_order )

**DESCRIPTION**

The CUTEST\_csetup\_threaded subroutine sets up the correct data structures for subsequent threaded computations on the problem decoded from a SIF file by the script *sifdecoder*. The problem under consideration is to minimize or maximize an objective function  $f(x)$  over all  $x \in R^n$  subject to general equations  $c_i(x) = 0$ , ( $i \in 1, \dots, m_E$ ), general inequalities  $c_i^l(x) \leq c_i(x) \leq c_i^u(x)$ , ( $i \in m_E + 1, \dots, m$ ), and simple bounds  $x^l \leq x \leq x^u$ . The objective function is group-partially separable and all constraint functions are partially separable.

**ARGUMENTS**

The arguments of CUTEST\_csetup\_threaded are as follows

**status** [out] - integer

the output status: 0 for a successful call, 1 for an array allocation/deallocation error, 2 for an array bound error, 3 for an evaluation error, 4 for an out-of-range thread,

**input** [in] - integer

the unit number for the decoded data; the unit from which OUTSDIF.d is read,

**out** [in] - integer

the unit number for any error messages,

**threads** [in] - integer

the total number of independent evaluation threads that are required,

**IO\_BUFFERS** [in] - integer

an array of different unit numbers, one entry for each thread, for any internal input/output,

**n** [inout] - integer

on input, the declared dimensions of X, X\_l and X\_u (see argument n in CUTEST\_cdimen). On output, the number of variables for the problem,

**m** [inout] - integer

on input, the declared dimensions of Y, C\_l, C\_u, EQUATN and LINEAR (see argument m in CUTEST\_cdimen). On output, the total number of general constraints,

**X** [out] - real/double precision

an array that gives the initial estimate of the solution of the problem,

**X\_l** [out] - real/double precision

an array that gives lower bounds on the variables,

**X\_u** [out] - real/double precision

an array that gives upper bounds on the variables,

**Y** [out] - real/double precision

an array that gives the initial estimate of the Lagrange multipliers at the solution of the problem. By convention, the signs of the Lagrange multipliers Y are set so the Lagrangian function can be written as  $l(x, y) = f(x) + y^T c(x)$ ,

**C\_l** [out] - real/double precision

an array that gives lower bounds on the inequality constraints,

**C\_u** [out] - real/double precision

an array that gives upper bounds on the inequality constraints,

**EQUATN** [out] - logical

a logical array whose i-th component is `.TRUE.` if the i-th constraint is an equation (i in E) and `.FALSE.` if the constraint is an inequality (i in I),

**LINEAR** [out] - logical

a logical array whose i-th component is `.TRUE.` if the i-th constraint is linear or affine and `.FALSE.` otherwise,

**e\_order** [in] - integer

if the user wishes the general equations to occur before the general inequalities in the list of constraints, `e_order` must be set to 1. If the general equations should follow the general inequalities, `e_order` must be set to 2. If the order is unimportant, `e_order` should be set to 0; any value except 1 and 2 will be interpreted as 0,

**l\_order** [in] - integer

if the user wishes the general linear (or affine) constraints to occur before the general nonlinear ones in the list of constraints, `l_order` must be set to 1. If the general linear constraints should follow the general nonlinear ones, `l_order` must be set to 2. If the order is unimportant, `l_order` should be set to 0; any value except 1 and 2 will be interpreted as 0,

**v\_order** [in] - integer

if the user wishes the nonlinear variables to occur before those that only appear linearly in the problem, in the list of variables, `v_order` must be set to 1; within the nonlinear variables the smaller set of either the nonlinear objective or nonlinear Jacobian variables will appear first. If the nonlinear variables must follow the linear ones, `v_order` should be set to 2. If the order is unimportant, `v_order` should be set to 0; any value except 1 and 2 will be interpreted as 0.

**APPLICATION USAGE**

A call to `CUTEst_csetup_threaded` must precede calls to other threaded evaluation tools, except `CUTEst_cdimen`, for generally-constrained problems.

**AUTHORS**

I. Bongartz, A.R. Conn, N.I.M. Gould, D. Orban and Ph.L. Toint

**SEE ALSO**

*CUTEst: a Constrained and Unconstrained Testing Environment with safe threads*,  
N.I.M. Gould, D. Orban and Ph.L. Toint,  
Technical Report, Rutherford Appleton Laboratory, 2013.

*CUTEr (and SifDec): A Constrained and Unconstrained Testing Environment, revisited*,  
N.I.M. Gould, D. Orban and Ph.L. Toint,  
ACM TOMS, **29**:4, pp.373-394, 2003.

*CUTE: Constrained and Unconstrained Testing Environment*, I. Bongartz, A.R. Conn, N.I.M. Gould and Ph.L. Toint, ACM TOMS, **21**:1, pp.123-160, 1995.

`cutest_usetup_threaded(3M)`, `cutest_cdimen(3M)`, `sifdecoder(1)`.