9.2 Local Minimum of a Multivariate Function with Linear Constraints

A. Purpose

The subroutine DMLC01 finds a local minimum of a continuously differentiable function f of n variables. Variables may be subject to bounds, and linear equality and inequality constraints. The problem statement is

$$\text{Minimize } f(\mathbf{x}), \text{ subject to}$$
$$A\mathbf{x} \le \mathbf{b}, \text{ and} \quad (1)$$
$$XL_j \le x_j \le XU_j, \quad j = 1, ..., n \quad (2)$$

where A is an $m \times n$ matrix, $\mathbf{b}$ is an m-vector, and $\mathbf{x}$ is the n-vector to be determined. It can be specified that the first MEQ rows of the system $A\mathbf{x} \le \mathbf{b}$ are to be equality constraints. This subroutine is also applicable to an unconstrained problem as one can set $m = 0$ and set the contents of XL() and XU() to have large magnitudes.

B. Usage

B.1 Program Prototype, Double Precision

INTEGER N, M, MEQ, LDA, IPRINT, MXEVAL, LIW, LW, IW(LIW)

DOUBLE PRECISION A(LDA,≥N), **B**(≥M), **XL**(≥N), **XU**(≥N), **X**(≥N), **ACC, W**(LW)

EXTERNAL DMLCFG

Assign values to N, M, MEQ, A(,), B(), XL(), XU(), X(), ACC, IPRINT, MXEVAL, LIW, and LW.

CALL DMLC01 (DMLCFG, N, M, MEQ, A, LDA, B, XL, XU, X, ACC, IPRINT, MXEVAL, IW, LIW, W, LW)

Results are returned in X(), IW(), and W().

B.2 Argument Definitions

DMLCFG [in] The name of a subroutine provided by the user to compute the function value, $f(\mathbf{x})$, and optionally the gradient vector, $\mathbf{g}$, $g_j = \partial f / \partial x_j$, $j = 1$, ..., N, using the current $\mathbf{x}$-vector given in X(). It must have an interface of the form:

```
SUBROUTINE DMLCFG(N, X, F, G, HAVEG)
INTEGER N
DOUBLE PRECISION X(N), F, G(N)
LOGICAL HAVEG
```

DMLCFG must store the function value in F.

DMLCFG may either compute the gradient vector or let the package compute an estimate of the gradient vector by computing finite differences of function values. Performance will generally be more reliable if DMLCFG computes the gradient vector.

If DMLCFG computes the gradient vector it must store it in G(1:N) and set HAVEG = .true. If DMLCFG does not compute the gradient it must set HAVEG = .false. and must not store anything into G().

DMLCFG must not change the values of N or X().

In some applications DMLCFG may need additional data that may have been input by the user's main program. One way to handle this in Fortran 77 is by use of named COMMON blocks.

N [in] Number of components in the vector $\mathbf{x}$. Require N ≥ 1.

M [in] Number of general linear constraints, *i.e.*, number of rows in the matrix A and components in the vector $\mathbf{b}$. Require M ≥ 0.

MEQ [in] Specifies that rows 1 through MEQ of the system $A\mathbf{x} \le \mathbf{b}$ are to be interpreted as equality constraints. Require 0 ≤ MEQ ≤ M.

A(,) [in] Array containing the M×N matrix A. Even if M = 0 the array A(,) must be declared and given positive dimensions. This is a Fortran 77 language requirement.

LDA [in] The leading dimension of the array A(,). Require LDA ≥ max(1, M).

B() [in] Array containing the M-vector $\mathbf{b}$. Even if M = 0 the array B() must be declared and given a positive dimension. This is a Fortran 77 language requirement.

XL(), XU() [in] Lower and upper bounds on the variables. See Eq. (2) above. Require XL(j) ≤ XU(j) for $j = 1$, ..., N. To leave a solution component effectively unconstrained, set its lower bound to a negative number of large magnitude and its upper bound to a large positive number. Do not set these bounds to such large magnitudes that computation of the difference, XU(j)− XL(j), would overflow for some j. Setting XL(j) = XU(j) is permitted and has the effect of fixing x_j at the value XL(j).

X() [inout] The vector of variables of the optimization calculation. On entry X() must contain an initial estimate of $\mathbf{x}$. The subroutines can usually cope with

a poor estimate, and the initial $\mathbf{x}$ is not required to satisfy Eqs. (1) and (2). On return X() will contain the subroutine's best estimate of the solution vector $\mathbf{x}$.

ACC [in] A tolerance on the first order Kuhn-Tucker conditions that the solution must satisfy. See Eq. (4), page 4 and Remark 4 in Section C. Setting ACC = 0.0 is permitted and essentially means the user wants as much accuracy as possible on the host computer. The termination will then usually be with INFO = 2, meaning Eq. (4) could not be satisfied.

IPRINT [in] Contains five print control parameters, packed as follows:

$$\text{IPRINT} = \text{IPFREQ} \times 100 + \text{IPTOL} \times 8 + \text{IPFRST} \times 4 + \text{IPMORE} \times 2 + \text{IPLAST}$$

Iteration counting only begins after a first feasible $\mathbf{x}$ is found. The items to be printed are selected by IPMORE. The other parameters specify when to print. If any of IPFREQ, IPTOL, IPFRST, or IPLAST are nonzero, there will be an initial printing of N, M, MEQ, ACC, RELACC, and TOL.

IPFREQ is zero or positive. If positive, printing is done on iterations 0, IPFREQ, 2*IPFREQ, etc.

IPTOL = 0 or 1. 1 means to print the new TOL value and the standard items each time TOL is changed. TOL is described in Section D.

IPFRST = 0 or 1. 1 means to print on the first iteration, *i.e.* on iteration number 0.

IPMORE = 0 or 1. Items always printed are

ITERC — The iteration count.

NFVALS — The count of function evaluations, which is tested against MXEVAL.

F — The current value of $f(\mathbf{x})$.

TOL — See Section D.

$\|\rho\|$ — The Euclidean norm of RESKT(1:N). This is the quantity that will be compared with ACC for the main convergence test. See Section D.

X(1:N) — The current value of $\mathbf{x}$.

When IPMORE = 1 the package also prints G(1:N), RESKT(1:N), IACT(1:NACT), and PAR(1:NACT). These quantities are described in Section D.

IPLAST = 0 or 1. 1 means to print the final results, and the reason for termination.

MXEVAL [in] If positive, this sets an upper limit on the number of function evaluations. Gradient evaluations and function evaluations for estimating the gradient are not counted. If MXEVAL = 0, there is no upper limit.

IW() [work, out] Integer work space of length LIW. Also used to return status information. On return IW() contains information as follows:

IW(1) contains INFO. Indicates reason for termination as follows:

1 means successful termination. X() is feasible and the convergence test involving ACC is satisfied.

2 means X() is feasible and satisfaction of the ACC test seems to be prevented by the precision of arithmetic being used. This mode of termination will commonly occur if the user sets ACC = 0.

3 means X() is feasible but the objective function fails to decrease, although a decrease is predicted by the current gradient vector. This may be due to limitation of computational precision as with INFO = 2, however, if the final RESKT() has components of large magnitude and the user has provided code to compute the gradient vector, this could be due to an error in the gradient code. One should also question the coding of the gradient when the final rate of convergence is slow.

4 means bad input values. See requirements on N, M, MEQ, XL(), XU(), LIW, and LW. No solution is computed in this case.

5 means the equality constraints are inconsistent. These constraints include any components of $\mathbf{x}$ that are frozen by setting $\text{XL}(j) = \text{XU}(j)$.

6 means the equality constraints and the bounds on the variables are inconsistent.

7 means there is no $\mathbf{x}$ satisfying all of the constraints. Specifically, when this return or an INFO = 6 return occurs, the current active constraints indexed in IACT(1:NACT) prevent any change in X() that reduces the sum of constraint violations.

8 means the limit set by MXEVAL has been reached, and there would have been further calculation otherwise.

9 means the solution is uniquely determined by the constraints, so there is no further freedom for minimization of $f(\mathbf{x})$. In this case the results returned in W() will include f evaluated at the solution, but will not include PAR() and RESKT(), and will include G() only if DMLCFG computes G().

IW(2) contains ITERC. Number of iterations used.

IW(3) contains NFVALS. Number of function evaluations, not counting extra function evaluations done to estimate the gradient when the gradient is not computed explicitly. In this latter case the actual number of function evaluations will be (K + 1) × NFVALS, where K is the number of solution components whose lower and upper bounds are not equal.

IW(4) contains NACT. The number of active constraints at the final **x**. Will be in the interval [MEQ, N].

IW(5:4+NACT) contains IACT(1:NACT). IACT() is used as work space of length M + 2× N. On return the first NACT locations contain the indices of the constraints that are active at the final point. Indices [1:M] refer to rows of the system $A\mathbf{x} \leq \mathbf{b}$. Indices [M+1:M+N] refer to component lower bounds. Indices [M+N+1:M+2×N] refer to component upper bounds.

LIW [in] Dimension of IW(). Require LIW ≥ 4 + M + 2 × N.

W() [work, out] Working space of floating point variables of length LW. Also used to return auxiliary results. On return W() contains information as follows:

W(1) contains FVAL, the final value of the objective function, f.

W(2) contains the Euclidean norm of the Kuhn-Tucker residual vector, *i.e.* $\|\rho\|$ as described in Section D.

W(3:2+N) contains the final gradient vector, G(1:N).

W(3+N:2+2×N) contains the Kuhn-Tucker residual vector, RESKT(1:N).

W(3+2×N:2+2×N+NACT) contains the Lagrange multipliers, PAR(1:NACT) where NACT has a value in [MEQ, N].

G(), RESKT(), and PAR() are described in Section D.

LW [in] Dimension of W(). Require LW ≥ 3 + M + N × (16 + N).

B.3 Modifications for Single Precision

For single precision usage change all subroutine names beginning with D, except D1MACH, to begin with S. Change the name D1MACH to R1MACH. Change all DOUBLE PRECISION type statements to REAL.

We suggest that the double precision version of this package be used, except on machines such as the Cray J90, where single precision arithmetic has precision of about 14.4 decimal places.

C. Examples and Remarks

Example: Minimize $f(\mathbf{x}) = \sum_{k=1}^{4} x_j \ln x_j$.

Let us bound each variable in [0,1] and require the sum of variables to be 1. With just these constraints a unique minimum would occur when $x_j = 0.25$, for all j. We shall add two more constraints to break up the symmetry:

$x_1 - x_2 = 0.25$

$x_2 - x_3 \geq 0.10$

Since inequality constraints must be expressed in less than or equal form, we rewrite this last constraint as

$-x_2 + x_3 \leq -0.10$

In matrix form the three general linear constraints are

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} 1.00 \\ 0.25 \\ -0.10 \end{bmatrix}$$

and we will specify that the first two rows are to be treated as equality constraints by setting MEQ = 2.

The program DRDMLC01 illustrates the use of DMLC01 to solve this problem, and also uses DCKDER to check the consistency of the function and gradient calculations. The output is shown in ODDMLC01. We have set ACC = 0.0, meaning we want as much accuracy as the host computer can provide. Note that the termination code is 2 indicating that the requested accuracy of 0.0 could not be attained. This is the usual termination code when the user has set ACC = 0.

We have set the lower bounds in XL() to 1.0D-6 rather than zero to avoid the possibility of the package requesting a function evaluation with some $x_j = 0$, since attempting to compute LOG(0.0D0) would cause an error stop. Alternatively this issue could be handled with appropriate logic in the function evaluation subroutine and XL() could be set to zero.

As print options we have selected printing only at the end (IPLAST = 1), and the more extensive number of printed items (IPMORE = 1). This is indicated by setting IPRINT = 2 × IPMORE + IPLAST = 3. Setting MXEVAL = 0 means we are not setting any limit on the number of function evaluations.

To illustrate the contents of IW() and W() on return we have printed the results from these arrays.

Remarks

1. It is important to the success of DMLC01 that the initial guess be as good as possible.

2. DMLC01 only finds a local minimum. Problems may have more than one local minimum, so caution in accepting results is suggested. It may be useful to solve the

problem several times using significantly different starting points.

3. Minimization of a nonlinear function is inherently difficult in many cases, and sometimes may require some interaction with the user. The intermediate output available from the subroutine may be useful if one has questions about the performance of the subroutine. It is not uncommon to make mistakes in writing the code for computing partial derivatives. This mistake is likely to cause a return with IW(1) = 3. The user can use the subroutine DCKDER of Chapter 8.3 to check the consistency of code for function and gradient evaluation.

4. It is reasonable to set ACC = 0.0 the first time this subroutine is applied to a new mathematical model. If one intends to solve more problems of similar form and wishes to reduce the number of iterations, one could turn on the intermediate output and try to determine a nonzero value of ACC that gives a result of adequate accuracy in fewer iterations. From Eq. (3) one sees that an appropriate nonzero value of ACC depends on the norms of $\mathbf{g}$ and of the row vectors of A, and the constraints active at the solution. If $\mathbf{g}$ is computed by finite-differences the magnitude of f must be considered also.

D. Functional Description

The algorithm provides for the bounds on the variables to be specified and treated separately from the general linear constraints because this is convenient for the user and allows for efficiencies in storage and execution time. However, for analysis of the problem it is more convenient to treat the bounds as additional general linear constraints. Thus the full set of constraints can be expressed as

$$\begin{bmatrix} A \\ -I \\ I \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} \mathbf{b} \\ -XL \\ XU \end{bmatrix}$$

with the first MEQ rows to be treated as equality constraints. Here I denotes the N × N identity matrix, XL denotes the N-vector of lower bounds, and XU denotes the N-vector of upper bounds.

Let C denote the left-side matrix and $\mathbf{d}$ the right-side vector in the above expression. Thus the constraints can be written simply as $C\mathbf{x} \leq \mathbf{d}$, where C and $\mathbf{d}$ each have M + 2N rows. Let $\mathbf{c}_i$ denote the column vector which is the transpose of row i of C.

If $XL_j = XU_j$ for some j, then row M+j of $C\mathbf{x} \leq \mathbf{d}$ will be treated as an equality constraint and row M + N + j will be ignored (not changing the indexing of other rows). A set $\mathcal{C}$ of linearly independent equality constraints is identified. This will generally consist of the constraints due to equality of lower and upper bounds, plus the first MEQ rows. However, it may consist of a proper subset of these if this set is not linearly independent. An internal variable, MEQL, is set to the number of these constraints and their indices are stored in IACT(1:MEQL). After setting MEQL and the contents of IACT(1:MEQL) these will remain unchanged throughout the rest of the algorithm.

A vector $\mathbf{x}$ is feasible if it satisfies $C\mathbf{x} \leq \mathbf{d}$, and in addition achieves equality for the rows in the set $\mathcal{C}$. With any feasible $\mathbf{x}$ the algorithm considers, it associates a list of indices of at most N of the rows of $C\mathbf{x} \leq \mathbf{d}$ that are satisfied with equality. These rows are called the active constraints (for $\mathbf{x}$) and the indices are denoted by IACT(k), k = 1, ..., NACT, where NACT depends on $\mathbf{x}$, but will satisfy MEQL ≤ NACT ≤ N.

The first-order Kuhn-Tucker necessary conditions for a solution $\mathbf{x}$ are that $\mathbf{x}$ must be feasible and that the gradient, $\mathbf{g}$, of f at $\mathbf{x}$ must be a linear combination of vectors $\mathbf{c}_i$ in the active set for $\mathbf{x}$, with combining coefficients (called Lagrange coefficients) that are of unrestricted sign for constraints in set $\mathcal{C}$, and are nonpositive for constraints not in set $\mathcal{C}$. Thus define the *Kuhn-Tucker residual vector*

$$\rho = \mathbf{g} - \sum_{k=1}^{NACT} \lambda_k \mathbf{c}_{IACT(k)}, \text{ with } \lambda_k \leq 0 \text{ for } k > \text{MEQL} \quad (3)$$

Then the Kuhn-Tucker condition on a vector $\mathbf{x}$ can be stated as the requirement that there exist λ_k's such that ρ is a zero vector. The convergence test used in the package is

$$\|\rho\| \leq \text{ACC} \quad (4)$$

where $\|\cdot\|$ denotes the Euclidean vector norm.

Internally the package stores $\mathbf{g}$ in G(), ρ in RESKT(), and the λ_k's in PAR(). These can be printed using the IPRINT argument and are returned in the W() array.

The package also uses internal tolerance parameters, RELACC and TOL. RELACC is set to about 100 times the machine precision. TOL is initially set to 0.01, and is reduced as the algorithm progresses until it reaches the value RELACC. TOL is used as a relative tolerance in checking the satisfaction of constraints. The technique of starting with TOL fairly large and later reducing it is a unique design feature of this algorithm. It has the effect of avoiding many small changes to $\mathbf{x}$ in the early stages of the algorithm.

The algorithm is described in detail by the author in [1] and [2]. He characterizes the algorithm as an active set method as described in [3], using BFGS updating of second derivative approximations as described in [4] and the matrix factorizations of [5]. Examples from [6] were used by the author in testing the package.

References

1. Michael J. D. Powell, **TOLMIN: A Fortran package for linearly constrained optimization calculations**. Technical Report DAMTP 1989/NA2, Department of Applied Mathematics and Theoretical Physics, University of Cambridge (June 1989) 98 pages.

2. M. J. D. Powell, *A tolerant algorithm for linearly constrained optimization calculations*, **Mathematical Programming 45**, 3, Ser. B (1988) 547–566. Appeared originally as DAMTP/1988/NA17, University of Cambridge, 1988.

3. Philip E. Gill, Walter Murray, and Margaret H. Wright, **Practical Optimization**, Academic Press, New York (1981) 401 pages. Sixth printing, 1987.

4. M. J. D. Powell, *Updating conjugate directions by the BFGS formula*, **Mathematical Programming 38**, 1 (1987) 20–46.

5. D. Goldfarb and A. Idnani, *A numerically stable dual method for solving strictly convex quadratic programs*, **Mathematical Programming 27**, 1 (1983) 1–33.

6. W. Hock and K. Schittkowski, **Test Examples for Nonlinear Programming Codes**, *Lecture Notes in Economics and Mathematical Systems 187*, Springer Verlag, Berlin (1981).

E. Error Procedures and Restrictions

On all returns, successful or not, the reason for the return is indicated by INFO, stored in IW(1). See the specification of IW(1) in Section B for the interpretation of these values. On returns with IW(1) from 3 to 8 an error message will be printed using the error message printing subroutines of Chapter 19.2 with an error level of 0.

F. Supporting Information

The source language is ANSI Fortran 77.

Entry	Required Files
DMLC01	AMACH, DERV1, DMLC, ERFIN, ERMOR, ERMSG, IERM1, IERV1
SMLC01	AMACH, ERFIN, ERMOR, ERMSG, IERM1, IERV1, SERV1, SMLC

Cvonerted from Powell's code cited above by C. L. Lawson, April, 1990 (with minor contribution from F. T. Krogh).

DRDMLC01

```
c     program DRDMLC01
c>> 2001-05-22 DRDMLC01 Krogh Minor change for making .f90 version.
c>> 1996-07-08 DRDMLC01 Krogh  Minor format change for C conversion.
c>> 1994-11-02 DRDMLC01 Krogh  Changes to use M77CON
c>> 1994-09-13 DRDMLC01 CLL
c>> 1992-05-13 CLL
c>> 1992-04-15 CLL
c>> 1992-01-21 CLL
c>> 1991-06-10 CLL
c>> 1992-01-16 CLL
c>> 1990-07-12 CLL
c     Demo driver for DMLC01.  Also uses DCKDER to check gradient.
c     Minimization with linear constraints.
c     The constraints are Ax .le. b with equality required in the first
c     two rows.
c     ------------------------------------------------------------------
c--D replaces "?": DR?MLC01, ?MLC01, ?XLOGX, ?CKDER
c     ------------------------------------------------------------------
      external DXLOGX
      integer MMAX, NMAX, LIW, LW
      parameter(MMAX = 3, NMAX = 4)
      parameter(LIW = 4 + MMAX + 2*NMAX)
      parameter(LW  = 3 + MMAX + NMAX*(16 + NMAX))
      integer I, IMAX, IPRINT, IW(LIW), J, JMAX
      integer M, MEQ, MODE, MXEVAL, N
      logical HAVEG
      double precision A(MMAX, NMAX), ACC, B(MMAX), FVAL(1), GRAD(NMAX)
      double precision GDUMMY(NMAX), TEST(NMAX), TSTMAX, W(LW)
```

```
      double precision X(NMAX), XL(NMAX), XU(NMAX)
      data M, MEQ, N / 3, 2, 4 /
      data ACC, IPRINT, MXEVAL / 0.0D0, 3, 0 /
      data ((A(I,J),J=1,4),I=1,3) / 1.0D0,  1.0D0, 1.0D0, 1.0D0 ,
     *                              1.0D0, -1.0D0, 0.0D0, 0.0D0,
     *                              0.0D0, -1.0D0, 1.0D0, 0.0D0 /
      data (B(I),I=1,3) / 1.0D0, 0.25D0, -0.1D0 /
      data (XL(J),J=1,4) / 4*1.0D-6 /
      data (XU(J),J=1,4) / 4*1.0D0 /
      data (X(J),J=1,4) / 0.7d0, 0.6d0, 0.5d0, 0.4d0 /
c     ------------------------------------------------------------------
      print*,
     * 'Program DRDMLC01..  Demo driver for DMLC01.  Also uses DCKDER.'
c
c                    Using DCKDER to check the gradient calculation.
c
      MODE = 1
      call DXLOGX( N, X, FVAL(1), GRAD, HAVEG)
   10 continue
      call DCKDER(MODE, 1, N, X, FVAL, GRAD, 1, TEST, IMAX, JMAX,TSTMAX)
         call DXLOGX( N, X, FVAL(1), GDUMMY, HAVEG)
      if(MODE .eq. 2) go to 10
      print'(/'' Using DCKDER to check the gradient calculation.'')'
      print'(/11x,''X(J) ='',5g11.3:/(17x,5g11.3))',(X(J),J=1,N)
      print'(11x,''FVAL ='',5g11.3)',FVAL(1)
      print'(7x,''Gradient ='',5g11.3:/(17x,5g11.3))',(GRAD(J),J=1,N)
      print'(9x,''TEST() ='',5g11.3/(17x,5g11.3))',(TEST(J),J=1,N)
      print'(11x,''JMAX ='',i3,'',           TSTMAX ='',g11.3)',JMAX,
     *     TSTMAX
c
c                Using DMLC01 to solve the minimization problem.
c
      print'(/'' Using DMLC01 to solve the minimization problem.'')'
      call DMLC01 (DXLOGX, N, M, MEQ, A, MMAX, B, XL, XU, X, ACC,
     *             IPRINT, MXEVAL, IW, LIW, W, LW)
      stop
      end
c     ==================================================================
      subroutine DXLOGX( N, X, F, G, HAVEG)
c     ------------------------------------------------------------------
      integer I, N
      double precision F, G(N), X(N)
      logical HAVEG
c     ------------------------------------------------------------------
      HAVEG = .true.
      F = 0.0D0
      do 10 I = 1,N
         F = F + X(I) * log(X(I))
         G(I) = 1.0D0 + log(X(I))
   10 continue
      return
      end
```

ODDMLC01

```
Program DRDMLC01..  Demo driver for DMLC01.  Also uses DCKDER.

Using DCKDER to check the gradient calculation.

          X(J) =  0.700       0.600       0.500       0.400
          FVAL =  -1.27
      Gradient =  0.643       0.489       0.307       0.837E-01
       TEST() = -0.425E-11 -0.507E-11 -0.279E-12  0.318E-10
          JMAX =  4,           TSTMAX =  0.318E-10

Using DMLC01 to solve the minimization problem.

     Beginning subroutine DMLC02, called from DMLC01:
     N =   4,  M =   3,  MEQ =   2,  ACC =   0.00
     RELACC =  0.222E-13,  TOL =   0.100E-01
[2]  DMLC01 quitting due to limitation of computational precision.

     6 iterations.      17 function evals.     F =   -1.2608
   TOL =  0.22204E-13   Norm of RESKT =  0.11342E-10
     X =  0.46004       0.21004       0.11004       0.21989
     G =  0.22355      -0.56047       -1.2069      -0.51462
 RESKT = -0.32740E-11 -0.32740E-11 -0.32740E-11  0.98221E-11
  IACT =    1     2     3
   PAR = -0.51462       0.73817      -0.69232
```