MINLPB4: A Fortran Subroutine for Mixed-Integer Nonlinear Optimization by Branch-and-Bound¹
- User’s Guide -

Oliver Exler, Thomas Lehmann, Klaus Schittkowski

Address: Department of Computer Science
University of Bayreuth
D-95440 Bayreuth

Date: July, 2010

Abstract

The Fortran subroutine MINLPB4 is a solver for mixed-integer nonlinear programming problems. It combines a branch-and-bound algorithm with the mixed-integer trust region method MISQP. Branching can either be applied to binary variables only or to binary and integer variables by applying MISQP as a continuous solver. Application is not restricted to convex problems. The usage of MINLPB4 is documented and some numerical results are presented.

Keywords: branch-and-bound; mixed-integer nonlinear programming; MINLP; numerical algorithms

¹Sponsored by Shell GameChanger, SIEP Rijswijk, under project number 4600003917
1 Introduction

The code MINLPB4 is designed to solve the mixed-integer nonlinear program (MINLP)

\[
\begin{align*}
\min & \quad f(x, b, y) \\
\text{s.t.} & \quad g_j(x, b, y) = 0, \quad j = 1, \ldots, m_e, \\
& \quad g_j(x, b, y) \geq 0, \quad j = m_e + 1, \ldots, m, \\
& \quad x_l \leq x \leq x_u, \\
& \quad y_l \leq y \leq y_u.
\end{align*}
\]

via branch-and-bound in combination with the mixed-integer nonlinear solver MISQP, see Exler et al. [4]. \(m_e\) is the number of equality constraints, \(m\) denotes the total number of constraints, \(x_l\) and \(x_u\) define box constraints for the continuous variables \(x \in \mathbb{R}^{n_c}\) and \(y_l\) and \(y_u\) are box constraints for the integer variables \(y \in \mathbb{N}^{n_i}\). \(B\) denotes the set \(\{0, 1\}\).

The user may choose between two different solution strategies. One consists of solving (1) by branching subject to the binary variables only. On the other hand, it is possible to branch on both binary and integer variables, and to use MISQP as a continuous nonlinear solver. The branch-and-bound process is performed by the code BFOUR of Lehmann et al. [7]. Depending on the chosen alternative, the resulting subproblems are either mixed-integer nonlinear programs

\[
\begin{align*}
\min & \quad f(x, b, y) \\
\text{s.t.} & \quad g_j(x, b, y) = 0, \quad j = 1, \ldots, m_e, \\
& \quad g_j(x, b, y) \geq 0, \quad j = m_e + 1, \ldots, m, \\
& \quad x_l \leq x \leq x_u, \\
& \quad b_l \leq b \leq b_u, \\
& \quad y_l \leq y \leq y_u
\end{align*}
\]

or continuous nonlinear problems

\[
\begin{align*}
\min & \quad f(x, b, y) \\
\text{s.t.} & \quad g_j(x, b, y) = 0, \quad j = 1, \ldots, m_e, \\
& \quad g_j(x, b, y) \geq 0, \quad j = m_e + 1, \ldots, m, \\
& \quad x_l \leq x \leq x_u, \\
& \quad b_l' \leq b \leq b_u', \\
& \quad y_l' \leq y \leq y_u'
\end{align*}
\]
Here, \( b'_l \) and \( b'_u \) are suitable bounds chosen by the branch-and-bound method, i.e., every component of both \( b'_l \) and \( b'_u \) may be either zero or one. \( y'_l \) and \( y'_u \) are box constraints within the original range \( \{y_l, y_u\} \) modified by the branch-and-bound procedure.

The mixed-integer programs must either be relaxable subject to the binary variables only or subject to all binary and integer variables depending on the branching strategy. In both cases, functions \( f(x, b, y) \) and \( g_i(x, b, y) \) must be continuously differentiable subject to the continuous and relaxed variables in order to apply efficient solution methods.

The relaxed problem (3) is solved by the continuous version of the nonlinear programming code MISQP of Exler et al. [3, 5]. For solving (2), MISQP is applied as a mixed-integer nonlinear solver.

Numerical experiments are reported in Exler et al. [4] which demonstrate the performance of the algorithm. They are based on a collection of mixed-integer test problems, which have been used in the past to develop and test mixed-integer programming algorithms, see Schittkowski [10]. Many of them are taken from the MINLPlib of Bussieck et al. [1].

The subsequent section introduces the basic concept of the branch-and-bound method and outlines some solution strategies. Implementation details and program documentation are found in Section 3, and an illustrative example is listed in Section 4.

2 The Branch-and-Bound Procedure

A branch-and-bound method is an iterative interaction of a branching and bounding process and an appropriate subproblem solver. Starting point is a relaxation of the original mixed-integer program, e.g., the corresponding problem obtained by replacing the condition \( y \in \mathbb{N}^n \) by \( y \in \mathbb{R}^n \) and \( b \in \mathbb{B}^n \) by \( b \in \mathbb{R}^n \) for some or all integer and binary variables. The relaxed problem must be solved by a suitable subproblem solver and is interpreted as the root node of a binary search tree. New nodes of the enumeration tree are successively generated by changing bounds for relaxed integer and/or binary variables. Subproblems are either continuous or mixed-integer programs possessing less integer or binary variables than the original problem. See also an early paper about a branch-and-bound algorithm for linear programming published by Dakin [2].

After exploring all possible nodes of the tree, i.e., if all corresponding subproblems are solved, the integer feasible subproblem solution possessing the lowest objective value is the optimal solution of the original problem (1).

In the following, we assume that the relaxations are convex, i.e., the continuous nonlinear programs (3) and the mixed-integer nonlinear program (2) are convex for all possible bounds. If the optimal solution of the relaxed root problem is integer, the branch-and-bound process can be stopped and the optimal solution is found. Otherwise, a fractional integer variable is selected and two different subproblems are generated. They are obtained by rounding the fractional value of a selected integer variable, say \( y_k \) which might
also be binary, to get two separate subproblems, one with upper bound $\lfloor y_k \rfloor$, another one with lower bound $\lfloor y_k \rfloor + 1$. Each subproblem determines a new child node of the binary search tree. An integer feasible solution of a subproblem gives an upper bound for the optimal objective function value of (1). The minimal objective function value at all nodes that are not completely explored, is a lower bound for the optimal objective value. Thus, whenever an feasible integer solution is found, there exists an upper and a lower bound for the optimal solution.

Furthermore, certain subtrees or nodes of the binary search tree can be eliminated or fathomed at an early stage:

- A subproblem is infeasible: All further subproblems obtained by branching from this node would also be infeasible, i.e., the node can be eliminated.

- The subproblem has a feasible integer solution: The corresponding optimal objective function value, if less than the known upper bound, provides a better upper bound for the optimal solution. Further branching from this node is not required.

- The continuous solution of a subproblem has a minimal solution value greater than the actual upper bound: Further branching from the node would only increase function values, and there is no chance to find a better integer solution. The node is eliminated.

Fathoming is in general critical for nonconvex problems, since it might cut off the optimal solution. The procedure is continued until all free nodes are either explored or fathomed. The integer feasible node with minimal value represents the solution of the mixed-integer problem or we get the information that no feasible mixed-integer solution exists.

There are two important steps by which the performance of a branch-and-bound algorithm is heavily influenced, the selection of a branching variable and the choice of the subsequent node of the enumeration tree to be explored next. The following options are implemented:

1. Selection of an integer variable with a fractional solution value for branching:

   (a) maximal fractional branching: Select an integer variable value from the solution of the relaxed subproblem with largest distance from nearest integer value.

   (b) minimal fractional branching: As above, with smallest distance from nearest integer value.

2. Search for a free node from where branching, i.e., the generation of two new subproblems, is started:

   (a) best of two: The optimal objective function values of the two child nodes are compared and the node with a lower value is chosen. If both are leafs, i.e., if the
corresponding solution is integral, or if the corresponding problem is infeasible or if there is already a better integral solution, strategy best of all is started. Warmstarts are possible, and memory is reduced.

(b) best of all: All unexplored nodes are compared and the node with lowest objective function value is selected. Quite often, large search trees are generated.

(c) depth first: Select a child node whenever possible. If the node is a leaf, the best of all strategy is applied. Warmstarts are profitable, memory is further reduced, but typically more relaxed subproblems must be solved.

Our implementation supports the input of an integer feasible point when starting the branch-and-bound process. This integral solution might either be known in advance or might be found by some primal heuristics. Depending on the distance of the corresponding objective function value from the optimal value, a solution is usually found much faster.

There are additional options which allow the user to influence the branch-and-bound process on a more technical level, which are not described in detail. In the nonconvex case, the fathoming procedure is adapted to reduce the probability of a cut-off of the optimal solution. In most cases, much more subproblems need to be solved.

The branch-and-bound procedure is especially valuable if some integer variables are categorial, i.e., enumerate internal states. A change of a categorial integer variable might lead to a completely different response of the model functions, and often correspond to binary variables. Thus, we allow branching subject to all integer variables or only to binary variables.

The resulting nonlinear mixed-integer or continuous subproblems are solved by a modified trust-region SQP code called MISQP, see Exler et al. [4]. The main assumption is that the integer variables exhibit a smooth influence on the model functions. It is supposed that the function values do not change drastically when an integer value is in- or decremented. Relaxable or concave functions are not required in contrast to most other available codes.

MISQP is stabilized by a trust region method in combination with second order corrections proposed by Yuan [12]. The Hessian of the Lagrangian function is approximated by a quasi-Newton update formula subject to the continuous and integer variables. But even in the convex case, convergence cannot be ensured. In the absence of integer variables, MISQP is a SQP trust region method suitable for solving continuous nonlinear programs. For more details, see Exler et al. [3, 5].

MISQP requires an efficient mixed-integer quadratic programming solver. The branch-and-cut code MIQL of see Lehmann et al. [8] is implemented, which uses the solver QL of Schittkowski [9] for solving a large number of continuous quadratic programs. The objective function matrix is a quasi-Newton BFGS matrix which is set up in the SQP code MISQP.
3 Program Documentation

MINLPB4 is implemented in form of a Fortran subroutine. The nonlinear mixed-integer subproblem is solved by the code MISQP of Exler et al. [5], the underlying mixed-integer quadratic programming problem by the code MIQL of Lehmann [8], an implementation of a branch-and-cut method based on the primal-dual method of Goldfarb and Idnani [6] implemented under the name QL, see Schittkowski [9], and the branch-and-bound code BFOUR of Lehmann et al. [7] is applied on top of the algorithm. Model functions and gradients must be provided by reverse communication. The user has to evaluate function and gradient values in the same program which executes MINLPB4, according to the following rules:

1. Choose starting values for the variables to be optimized, and store them in X, first the continuous, then the binary followed by the integer variables. Do not forget to initialize ROPT, IOPT, and LOPT.

2. Compute objective and all constraint function values values at X and store them in F and G, respectively.

3. Compute gradients of objective function and all constraints, and store them in DF and DG, respectively. The $j$-th row of DG contains the gradient of the $j$-th constraint, $j=1,...,m$. If branched only on binary variables, it is sufficient to determine the gradients subject to the continuous and binary variables and the integer variables specified in IDERIV. Otherwise, gradients subject to all variables need to be provided.

4. Set IFAIL=0 and execute MINLPB4.

5. If MINLPB4 terminates with IFAIL=0, the internal stopping criteria are satisfied.

6. In case of IFAIL>0, an error occurred.

7. If MINLPB4 returns with IFAIL=-1, compute objective function values and constraint values for all variables found in X, store them in F and G, and call MINLPB4 again.

8. If MINLPB4 terminates with IFAIL=-2, compute gradient values subject to variables stored in X, and store them in DF and DG. If branched only on binary variables, it is sufficient to determine the gradients subject to the continuous and binary variables and the integer variables specified in IDERIV. Otherwise, gradients subject to all variables need to be provided. Then call MINLPB4 again.

9. If MINLPB4 terminates with IFAIL=-3, compute function and gradient values subject to variables stored in X, and store them in F, G, DF, and DG. If branched
only on binary variables, it is sufficient to determine the gradients subject to the continuous and binary variables and the integer variables specified in IDERIV. Otherwise, gradients subject to all variables need to be provided. Then call MINLPB4 again.

Note that derivatives subject to integer variables are approximated by MINLPB4 internally using a difference formula, if MINLPB4 only branches on binary variables. The calling sequence and the meaning of the parameters of MINLPB4 are described below, where default values, as far as applicable, are set in brackets. In some cases, we also present alternative identifiers for parameters of option arrays for compatibility with other related software documentations.

MINLPB4 must be linked with the calling routine of the user, the mixed-integer nonlinear programming code MISQP, the branch-and-bound code BFOUR, the Cholesky decomposition code CHOLKY, and the mixed-integer quadratic optimization code MIQL.

**Usage:**

```fortran
CALL MINLPB4( M, ME, MMAX, N, NBIN,
/ MINT, X, F, G, DF,
/ DG, XL, XU, ACC, MAXIT,
/ MAXNDE, IPRINT, IOUT, IFAIL, IDERIV,
/ ROPT, IOPT, LOPT, RW, LRW,
/ IW, LIW, LW, LLW )
```

**Parameter Definition:**

- **M**: Input parameter defining the total number of constraints.
- **ME**: Input parameter defining the number of equality constraints.
- **MMAX**: Row dimension of array DG containing Jacobian of constraints. MMAX must be at least one and greater or equal to M.
- **N**: Input parameter defining the total number of optimization variables, continuous, integer, and boolean ones.
- **NBIN**: Input parameter defining the number of binary optimization variables, must be less than or equal to N.
NINT: Input parameter for the number of non-binary integer variables, must be less than or equal to N.

X(N): Input and output vector containing the starting point at first call. On return, X is replaced by the current iterate. The first N-NBIN-NINT positions are assigned to continuous variables, the subsequent NBIN coefficients to binary variables, and the remaining NINT positions to non-boolean integer variables.

F: Input parameter containing the objective function value at the actual iterate X.

G(MMAX): Input vector containing the values of the constraints at the actual iterate X, first ME equality constraints, then M-ME inequality constraints.

DF(N): Input vector containing the values of the gradient of the objective function at the current iterate X. If branched only on binary variables (LOPT(24)=.TRUE.), it is sufficient to determine the gradients subject to the continuous and binary variables and the integer variables specified in IDERIV. Otherwise, gradients subject to all variables need to be provided. See also description of LOPT(24) and IDERIV below.

DG(MMAX,N): Input matrix containing the values of the Jacobian of the constraints at the current iterate X, first for the ME equality constraints, then for M-ME inequality constraints. In the driving program, the row dimension of DG has to be equal to MMAX. If branched only on binary variables (LOPT(24)=.TRUE.), it is sufficient to determine the gradients subject to the continuous and binary variables and the integer variables specified in IDERIV. Otherwise, gradients subject to all variables need to be provided. See also description of LOPT(24) and IDERIV below.

XL(N): Input and output vector containing the lower bounds of the variables X. Note that bounds of binary or integer variables, respectively, may be modified during the branch-and-bound process.

XU(N): Input and output vector containing the upper bounds, see XL.

ACC: Input parameter defining the tolerance for detecting integer values and for termination. If ACC is less than machine precision, then ACC is internally set to machine precision multiplied by 1,000.
MAXIT : Maximum number of iterations (100).

MAXNDE: Input parameter defining the maximum number of subproblems or tree nodes, respectively, that can be generated for solving mixed-integer nonlinear programs by MISQP. MAXNDE should be at least 1,000.

IPRINT: Input parameter defining the output level of MINLPB4:
0: No output.
1: Only final convergence analysis.
2: One line of intermediate output.
3: More detailed information for each iteration.
4: Complete output for debugging.

IOUT: Input parameter defining the desired output unit number, i.e., all write-statements start with 'WRITE(IOUT,... '.

IFAIL: Output parameter showing the reason for terminating a solution process. Initially IFAIL must be set to zero. On return, IFAIL contains one of the subsequent values:
-3: Compute new function and gradient values, store in F, G, DF and DG. See description of DF, DG, IDERIV and BINBB.
-2: Compute new gradient values, store in DF and DG. See description of DF, DG, IDERIV and BINBB.
-1: Compute new function values in F and G, see above.
0: Optimality conditions satisfied.
1: Integer solution found, but maximum number of nodes reached.
2: No integer solution exists.
3: No integer solution found within maximum number of nodes.
4: Original relaxed problem not solvable.
5: Problem not solvable due to error in MISQP.
6: Length of working array RW, IW, or LW too short.
7: One of the parameters N, M, ME, NINT, NBIN, or MNN incorrect.
8: IOUT or IPRINT incorrect.
9: Lower variable bound (XL) greater than upper variable bound (XU).
10: Input options incorrect.
11: Internal MISQP error.
12: Internal error of branch-and-bound routine.

IDERIV(NINT) : Logical input array specifying integer variables whose derivatives are always provided by the user and stored in DG and DF at the corresponding column positions:
IDERIV(I)=TRUE : Column NCONT+NBIN+I of DG and position NCONT+NBIN+I of DF are set by the user.
IDERIV(I)=FALSE : Column NCONT+NBIN+I of DG and position NCONT+NBIN+I of DF are set by MINLPB4 by evaluating functions at neighbored grid points.

Here we have NCONT = N - NINT - NBIN.
Note that this feature can only be applied for integer variables if MINLPB4 branches only on binary variables.

ROPT(60): Input vector defining the double precision options. To generate default values, all entries must be initialized with -1.0D0 before calling MINLPB4.

MISQP:
ROPT(1) Termination tolerance of the internal QP solver (ACCQP, 1.0E-12).
ROPT(2) Factor for increasing a penalty parameter, must be greater than one (11).
ROPT(3) Factor for increasing an internal descent parameter DELTA, see ROPT(5). ROPT(3) must be greater than one (11).
ROPT(4) Initial penalty parameter, greater than one (SIGMA, 10).
ROPT(5) Initial descent constant, smaller than one (DELTA, 0.01).
ROPT(6) Initial continuous trust region radius, greater than zero (10).
ROPT(7) Initial integer trust region radius, not less than one (10).

MINLPB4:
ROPT(22) Output parameter for the actual lower bound.
ROPT(23) Output parameter for the function value of the best known integer feasible solution.
IOPT(60): Input vector defining the integer options. To generate default values, all entries must be initialized with -1 before calling MINLPB4.

MISQP:
   IOPT(1) To enable cut generation of the QP solver, row dimension MMAX is enlarged internally by IOPT(1) rows (MAXCUT, 100).
   IOPT(2) Maximum number of successive iterations which are considered for the non-monotone trust region algorithm, influences length of working array, must be less than 100 (NONMON, 10).
   IOPT(3) Print level of the subproblem solver MIQL (2).
   IOPT(4) Output for number of gradient evaluations.
   IOPT(5) Output for number of function evaluations.

MINLPB4:
   IOPT(21): Input parameter for selecting a branching variable (1),
       1 : Maximal fractional branching.
       2 : Minimal fractional branching.
   IOPT(22): Input parameter for selecting the next node (1),
       1 : Best of all.
       2 : Best of two.
       3 : Depth first.
   IOPT(23): Maximal number of subproblems or tree nodes, respectively, of the outer branch-and-bound algorithm (MAXNBB, 10000).
   IOPT(26): Input parameter for selecting branching direction (1),
       0 : Branch left at the current node.
       1 : Branch right at the current node.

MIQL:
   IOPT(41) Branching rule (1):
       1 : Maximal fractional branching.
       2 : Minimal fractional branching.
   IOPT(42) Node selection strategy (3):
       1 : Best of all.
       2 : Best of two.
3: Depth first.

IOPT(43) Maximal number of nodes, should be at least \((2N+2M+6)\)\(^2\) to guarantee generation of disjunctive cuts, and must be at least \((2N+M+4)\)\(^2\) for recomputing Lagrange multipliers (MAXNBB, 100000).

IOPT(44) Maximal number of successive warmstarts to avoid numerical instabilities (100).

IOPT(45) Calculate improved bounds if best-of-all selection strategy is used (0).

IOPT(46) Select direction for depth-first according to value of Lagrange function (0).

IOPT(47) Cholesky decomposition mode (1),
   0: Calculate Cholesky decomposition once and reuse it.
   1: Calculate new decomposition if warmstart is not activated.

IOPT(48) Control the cutting process (0),
   0: No cuts.
   1: Disjunctive cuts only.
   2: Complemented mixed integer rounding (CMIR) cuts only.
   3: Both disjunctive and CMIR cuts.

IOPT(49) Maximal number of cycles for disjunctive cuts (1).

IOPT(50) Maximal number of cycles for CMIR cuts (1).

IOPT(51) Primal heuristic mode (0),
   0: No primal heuristics.
   1: Nearest integer.
   2: Feasibility pump.

LOPT(60): Input vector defining the logical options. All entries must be initialized with TRUE before calling MINLPB4.

MISQP:

LOPT(1) Additional restarts are performed to check whether an improvement of the search direction is possible. Restart is recommended for highly nonlinear constraints, especially for nonlinear equality constraints (RESOPT, .TRUE.).
LOPT(2) MISQP internally scales variables (SCALE, .TRUE.).
LOPT(3) MISQP modifies the Hessian approximation in order to get more accurate search directions. Calculation time is increased in case of a large number of integer variables (BMOD, .TRUE.).

MINLPB4:
LOPT(21) Output parameter indicating that a new best integer feasible solution found.
LOPT(22) Output parameter indicating that the current node is integral.
LOPT(23) Output parameter indicating that the current node is marked.
LOPT(24) Flag for branching only on binary variables (.TRUE.) or on integer and binary variables (.FALSE.). In the first case, it is sufficient to determine the gradients subject to the continuous and binary variables and the integer variables specified in IDERIV. Otherwise, gradients subject to all variables need to be provided (BINBB). See also description of DG, DF and IDERIV.
LOPT(25) Output parameter to indicate a known integer feasible solution.
LOPT(26) Flag for indicating a convex problem or not (CONVEX).

RW(LRW): Real working array of length LRW.
LRW: Input parameter defining the length of RW, must be at least
\[ 5N^2 + 2MMAX0N + 74N + 36MMAX0 + 3MAXNDE + 3MAXNBB + 700, \]
where MMAX0 = M + ME + MAX(NINT+NBIN,IOPT(1)).

IW(LIW): Integer array of length LIW.
LIW: Input parameter defining the length of IW, must be at least
\[ 14N + M + 5MMAX0 + 6MAXNDE + 6MAXNBB + 180. \]

LW(LLW): Logical working array of length LLW.
LLW: Input parameter defining the length of LW, must be at least
\[ 7N + MMAX0 + 200. \]

4 Example

To give an example how to organize the code, we consider a pseudo-convex test problem of Westerlund and Pörn [11], see also MITP17 in Schittkowski [10], with one continuous
and one integer variable,

$$
\begin{align*}
\min & \quad \frac{(x - 3)^2 - 10x}{3x + y + 1} \\
x \in \mathbb{R}, y \in \mathbb{N} & : \\
5y - (x - 7)^2 & \geq 0 \\
1.8y - x & \geq 0 \\
1 & \leq x \leq 8 \\
1 & \leq y \leq 8
\end{align*}
$$

(4)

The Fortran source code for executing MINLPB4 is listed below. Derivatives subject to the integer variable are evaluated analytically and the starting point is set to the lower bound.

```fortran
IMPLICIT NONE
INTEGER N, MMAX, MAXNDE, MAXNBB, MAXCUT, LRW, / LIW, LLW
PARAMETER (N = 2, / MMAX = 2, / MAXNDE = 1000, / MAXNBB = 500, / MAXCUT = 10, / MMAX0 = 2*MMAX + MAX0(N,MAXCUT), / LRW = 5*N*N/2 + 2*MMAX0*N + 74*N + 36*MMAX0 + 3*MAXNDE + 3*MAXNBB + 700, / LIW = 14*N + 5*MMAX0 + MMAX + 6*MAXNDE / + 6*MAXNBB + 180, / LLW = 7*N + MMAX0 + 200)
INTEGER IW(LIW), NINT, NBIN, NCONT, M, ME, IOUT, MAXIT, / IPRINT, IFAIL, I, J, IOPT(60)
DOUBLE PRECISION F, X(N), G(MMAX), DF(N), DG(MMAX,N), / XL(N), XU(N), ACC, RW(LRW), ROPT(60), / EPS, EPSREL, FBCK, GBCK(MMAX), XBCK
LOGICAL LW(LLW), LOPT(60), IDERIV(N)
C
C Set some constants and initial values
C
NINT = 1
NBIN = 0
NCONT = N - NINT - NBIN
M = 2
ME = 0
DO I=1,N
   XL(I) = 1.0D0
   X(I) = 1.0D0
   XU(I) = 8.0D0
END DO
DO I=1,60
   ROPT(I) = -1.0D0
   IOPT(I) = -1
   LOPT(I) = .TRUE.
END DO
C
C Tolerances for calling MINLPB4
C
ACC = 1.0D-10
MAXIT = 150
```
IOPT(23) = MAXNBB
LOPT(1) = .FALSE.
LOPT(3) = .FALSE.
IPRINT = 3
IOUT = 6
IDERIV(1) = .TRUE.

C

Optimization block (reverse communication)
C

IFAIL = 0
1 CONTINUE
C

This is the main block to compute all function and derivative values.
C

IF (IFAIL.NE.-2) THEN
  F = ((X(1) - 3.0D0)**2 - 10.0D0*X(1))
    /(3.0D0*X(1) + X(2) + 1.0D0)
  G(1) = 5.0D0*X(2) - (X(1) - 7.0D0)**2
  G(2) = 1.8D0*X(2) - X(1)
ENDIF

IF (IFAIL.EQ.0.OR.IFAIL.LT.-1) THEN
  DF(1) = (3.0D0*X(1)**2 + 2.0D0*X(1) - 16.0D0*X(2)
    + 2.0D0*X(1)*X(2) - 43.0D0)
    /(3.0D0*X(1) + X(2) + 1.0D0)**2
  DF(2) = -((X(1) - 3.0D0)**2 - 10.0D0*X(1))
    /(3.0D0*X(1) + X(2) + 1.0D0)**2
  DG(1,1) = -2.0D0*(X(1) - 7.0D0)
  DG(2,1) = -1.0D0
  DG(1,2) = 5.0D0
  DG(2,2) = 1.8D0
ENDIF

C

CALL MINLPB4( M, ME, MMAX, N, NBIN,
    NINT, X, F, G, DF,
    DG, XL, XU, ACC, MAXIT,
    MAXNDE, IPRINT, IOUT, IFAIL, IDERIV,
    ROPT, IOPT, LOPT, RW, LRW,
    IW, LIW, LW, LLW )

IF (IFAIL.LT.0) GOTO 1

C

End of main program
C

STOP
END

The following output should appear on screen:

START OF A NLP BASED BRANCH AND BOUND METHOD

Parameters:
  ACC = 0.100D-09
  MAXBB = 500
  IPRINT = 3
  NCONT = 1
  NBIN = 0
  NINT = 1
M = 2
ME = 0

START OF THE MIXED-INTEGER SQP CODE MISQP

Parameters:
ACC = 0.100D-09
ACCQP = 0.100D-11
MAXIT = 150
RESOPT = T
SCALE = T
BMOD = T
NONMON = 2
MAXDNS = 1000
IPRINT = 2
NCONT = 1
NBI = 0
NINT = 1
M = 2
ME = 0

Output in the following order:
IT - iteration number
F - objective function value
MCV - maximum constraint violation
SIGMA - penalty parameter
DMAXC - maximum norm of continuous step D_C
DMAXB - maximum norm of binary step D_B
DMAXI - maximum norm of integer step D_I
R - ratio actual/predicted reduction

<table>
<thead>
<tr>
<th>IT</th>
<th>F</th>
<th>MCV</th>
<th>SIGMA</th>
<th>DMAXC</th>
<th>DMAXB</th>
<th>DMAXI</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.1200000D+01</td>
<td>0.31D+02</td>
<td>0.10D+02</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
</tr>
<tr>
<td>1</td>
<td>-0.253111D+01</td>
<td>0.47D+01</td>
<td>0.10D+02</td>
<td>0.31D+01</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
</tr>
<tr>
<td>2</td>
<td>-0.241973D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.79D+00</td>
<td>0.00D+00</td>
<td>0.10D+01</td>
<td>0.86D+00</td>
</tr>
<tr>
<td>3</td>
<td>-0.242406D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.76D-01</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.10D+01</td>
</tr>
<tr>
<td>4</td>
<td>-0.243209D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.34D+00</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.15D+00</td>
</tr>
<tr>
<td>5</td>
<td>-0.244432D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.39D+00</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.11D+01</td>
</tr>
<tr>
<td>6</td>
<td>-0.244444D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.60D-01</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.11D+01</td>
</tr>
<tr>
<td>7</td>
<td>-0.244444D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.54D-02</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.10D+01</td>
</tr>
<tr>
<td>8</td>
<td>-0.244444D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.67D-04</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.10D+01</td>
</tr>
<tr>
<td>9</td>
<td>-0.259584D+01</td>
<td>0.23D+00</td>
<td>0.10D+02</td>
<td>0.71D+00</td>
<td>0.00D+00</td>
<td>0.10D+01</td>
<td>0.10D+01</td>
</tr>
<tr>
<td>10</td>
<td>-0.244444D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.72D+00</td>
<td>0.00D+00</td>
<td>0.10D+01</td>
<td>0.10D+01</td>
</tr>
<tr>
<td>11</td>
<td>-0.244444D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.30D-02</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.17D+01</td>
</tr>
<tr>
<td>12</td>
<td>-0.244444D+01</td>
<td>0.00D+00</td>
<td>0.10D+02</td>
<td>0.79D-02</td>
<td>0.00D+00</td>
<td>0.00D+00</td>
<td>0.10D+01</td>
</tr>
</tbody>
</table>

--- FINAL CONVERGENCE ANALYSIS ---

Objective function value: F(X) = -0.24444444D+01
Approximation of solution: X = 0.43333333D+01 0.30000000D+01
Approximation of multipliers: U = 0.00000000D+00 0.00000000D+00 0.00000000D+00 0.14377647D+00
0.00000000D+00 0.00000000D+00
Constraint function values: G(X) = 0.78888886D+01 0.10666667D+01
Distances from lower bounds: XL-X = -0.33333333D+01 -0.20000000D+01 -0.20000000D+01
Distances from upper bounds: \( X_{U} - X \) = 
\[ 0.36666667D+01 \quad 0.50000000D+01 \]
Number of function calls: \( N_{FUNC} \) = 13
- within TR method: \( N_{F_TR} \) = 13
- integer derivatives: \( N_{F_2D} \) = 0
Number of gradient calls: \( N_{GRAD} \) = 13
Number of calls of QP solver: \( N_{QL} \) = 14
- 2nd order corrections: \( N_{QL2} \) = 0
Number of B&B nodes: \( N_{NODES} \) = 39
Termination reason: \( IFAIL \) = 0

--- FINAL CONVERGENCE ANALYSIS ---

Objective function value: \( F(X) \) = -0.24444444D+01
Approximation of solution: \( X \) = 
\[ 0.43333333D+01 \quad 0.30000000D+01 \]
Constraint function values: \( G(X) \) = 
\[ 0.78888886D+01 \quad 0.10666667D+01 \]
Distances from lower bounds: \( X_{L} - X \) = 
\[ -0.33333333D+01 \quad -0.20000000D+01 \]
Distances from upper bounds: \( X_{U} - X \) = 
\[ 0.36666667D+01 \quad 0.50000000D+01 \]
Number of function calls: \( N_{FUNC} \) = 12
Number of gradient calls: \( N_{GRAD} \) = 12
Termination reason: \( IFAIL \) = 0

References


