MISQP: A Fortran Subroutine of a Trust Region SQP Algorithm for Mixed-Integer Nonlinear Programming[1]

- User's Guide -

Address: Prof. K. Schittkowski
Siedlerstr. 3
D - 95488 Eckersdorf
Germany

Phone: (+49) 921 32887

E-mail: klaus@schittkowski.de

Web: http://www.klaus-schittkowski.de

Date: May, 2013

Abstract

The Fortran subroutine MISQP solves mixed-integer nonlinear programming problems by a modified sequential quadratic programming (SQP) method. Under the assumption that integer variables have a *smooth* influence on the model functions, i.e., that function values do not change drastically when in- or decrementing an integer value, successive quadratic approximations are applied. The algorithm is stabilized by a trust region method including Yuan's second order corrections. It is not assumed that integer variables are relaxable. In other words, problem functions are evaluated only at integer points. The Hessian of the Lagrangian function is approximated by BFGS updates subject to the continuous and integer variables. The usage of the code is documented and illustrated by an example.

Keywords: mixed-integer nonlinear programming, sequential quadratic programming, SQP, trust region method

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

1 Introduction

We consider the general optimization problem to minimize an objective function f under nonlinear equality and inequality constraints,

$$\begin{aligned} & \min f(x,y) \\ & g_j(x,y) = 0\ , \quad j = 1,\ldots,m_e\ , \\ x \in \mathbb{R}^{n_c}, y \in \mathbb{N}^{n_i}: \ & g_j(x,y) \geq 0\ , \quad j = m_e+1,\ldots,m\ , \\ & x_l \leq x \leq x_u \qquad , \\ & y_l \leq y \leq y_u \qquad , \end{aligned} \tag{1}$$

where x and y denote the vectors of the continuous and integer variables, respectively. It is assumed that the problem functions $f(x,y)$ and $g_j(x,y)$, $j = 1, \ldots, m$, are continuously differentiable subject to all $x \in \mathbb{R}^{n_c}$.

Trust region methods have been invented many years ago first for unconstrained optimization, especially for least squares optimization, see for example Powell [25], or Moré [23]. Extensions were developed for non-smooth optimization, see Fletcher [13], and for constrained optimization, see Celis [7], Powell and Yuan [26], Byrd et al. [6], Toint [34] and many others. A comprehensive review on trust region methods is given by Conn, Gould, and Toint [8].

On the other hand, sequential quadratic programming or SQP methods belong to the most frequently used algorithms to solve practical optimization problems. The theoretical background is described e.g. in Stoer [33], Fletcher [12], or Gill et al. [16].

However, the situation becomes much more complex if additional integer variables must be taken into account. Numerous algorithms have been proposed in the past, see for example Floudas [15] or Grossmann and Kravanja [18] for review papers. Typically, these approaches require convex model functions and continuous relaxations of integer variables. By a continuous relaxation, we understand that integer variables can be treated as continuous variables, i.e., function values can also be computed between successive integer points. There are branch-and-bound methods where a series of relaxed nonlinear programs obtained by restricting the variable range of the relaxed integer variables must be solved, see Gupta and Ravindran [19] or Borchers and Mitchell [2]. When applying an SQP algorithm for solving a subproblem, it is possible to apply early branching, see also Leyffer [21]. Pattern search algorithms are available to search the integer space, see Audet and Dennis [1]. After replacing the integrality condition by continuous nonlinear constraints, it is possible to solve the resulting highly nonconvex program by a global optimization algorithm, see e.g. Li and Chou [22]. Outer approximation methods are investigated by Duran and Grossmann [9] and Fletcher and Leyffer [14], where a sequence of alternating mixed-integer linear and nonlinear programs must be solved. Alternatively, it is also possible to apply cutting planes similar to linear programming, see Westerlund and Pörn [35].

But fundamental assumptions are often violated in practice. Many real-life mixed-integer problems are not relaxable, and model functions are highly nonlinear and nonconvex. Moreover, some approaches require detection of infeasibility of nonlinear programs, a highly unattractive feature from the computational point of view. We assume now that integer variables are not relaxable, that there are relatively large ranges for integer values, and that the integer variables possess some kind of *physical* meaning, i.e., are *smooth* in a certain sense. It is supposed that a slight alteration of an integer value, say by one, changes the model functions only slightly, at least much less than a more drastic change. Typically, relaxable programs satisfy this requirement. In contrast to them, there are *categorical* variables which are introduced to enumerate certain situations and where any change leads to a completely different category and thus to a completely different response.

A practical situation is considered by Bünner and Schittkowski [3], where typical integer variables are the number of fingers and the number of layers of an electrical filter. By increasing or decreasing the number of fingers by one, we expect only a small alteration in the total response, the transmission energy. The more drastically the variation of the integer variable is, the more drastically model function values are supposed to change.

Thus, we propose an alternative idea in Section 2 where we try to approximate the Lagrangian subject to the continuous and integer variables by a quadratic function based on a quasi-Newton update formula. Instead of a line search as is often applied in the continuous case, we use trust regions to stabilize the algorithm and to enforce convergence following the continuous trust region method of Yuan [37] with second order corrections. The specific form of the quadratic programming subproblem avoids difficulties with inconsistent linearized constraints and leads to a convex mixed-integer quadratic programming problem, which can be solved by any available algorithm, for example, a branch-and-bound method. Algorithmic details and numerical results are reported in Exler et. al. [11].

The numerical subroutines are implemented in Fortran, as close as necessary to F77, without global variables (COMMON) or 'tricky' Fortran constructs like EQUIVALENCE or ENTRY. Transfer to C by f2c is possible. Nonlinear functions and, if required, partial derivatives must be provided by reverse communication, the most flexible and most user-friendly way. The usage of the Fortran subroutine is documented in Section 4, and Section 5 contains an illustrative example.

2 The Mixed-Integer Trust Region SQP Method

Basically, integer variables lead to extremely difficult optimization problems. Even if we assume that the integer variables are relaxable and that the resulting continuous problem is strictly convex, it is possible that the integer solution is not unique. There is no numerically applicable criterion to decide whether we are close to an optimal solution nor can we retrieve any information about the position of the optimal integer solution from

the continuous solution of the corresponding relaxed problem.

The basic idea of a trust region method is to compute a new iterate by a second order model or a close approximation, see Exler and Schittkowski [10] or Exler et. al. [11] for more details. The stepsize is restricted by a trust region radius Δ_k, where k denotes the k-th iteration step. Subsequently, a ratio r_k of the actual and the predicted improvement subject to a certain merit function is computed. The trust region radius is either enlarged or decreased depending on the deviation of r_k from the ideal value $r_k = 1$. If sufficiently close to a solution, the artificial bound Δ_k should not become active, so that the new trial proposed by the second order model can always be accepted.

For the continuous case, the superlinear convergence rate can be proved, see Fletcher [13], Burke [4], or Yuan [36]. The individual steps depend on a large number of constants, which are carefully selected based on numerical experience.

The goal is to apply a trust region SQP algorithm and to adapt it to the mixed-integer case with as few alterations as possible. Due to the integer variables, the quadratic programming subproblems that have to be solved are mixed-integer quadratic programming problems and can be solved by any available algorithm. Since the generated subproblems are always convex, we apply the branch-and-bound code MIQL of Lehmann et al. [20]. The mixed-integer quadratic programming problems to be solved, are of the form

$$
\begin{array}{ll}
 & \min \frac{1}{2} \begin{pmatrix} d \\ e \end{pmatrix}^T B_k \begin{pmatrix} d \\ e \end{pmatrix} + \nabla_x f(x_k, y_k)^T d + d_y f(x_k, y_k)^T e + \sigma_k \delta \\
 & -\delta \leq \nabla_x g_j(x_k, y_k)^T d + d_y g_j(x_k, y_k)^T e + g_j(x_k, y_k) \leq \delta \ , \quad j = 1, \ldots, m_e \ , \\
d \in \mathbb{R}^{n_c}, & \\
e \in \mathbb{N}^{n_i}, \ : & -\delta \leq \nabla_x g_j(x_k, y_k)^T d + d_y g_j(x_k, y_k)^T e + g_j(x_k, y_k) \ , \quad j = m_e + 1, \ldots, m \ , \\
\delta \in \mathbb{R} & \\
 & \max(x_l - x_k, -\Delta_k^c) \leq d \leq \min(x_u - x_k, \Delta_k^c) \ , \\
 & \max(y_l - y_k, -\Delta_k^i) \leq e \leq \min(y_u - y_k, \Delta_k^i) \ , \\
 & 0 \leq \delta \ .
\end{array}
\tag{2}
$$

The solution is denoted by d_k, e_k, and δ_k.

Since we do not assume that (1) is relaxable, i.e., that f and $g_1, \ldots, g_m$ can be evaluated at any fractional parts of the integer variables, we approximate the first derivatives at $f(x, y)$ by the difference formula

$$
d_y f(x, y) = \frac{1}{2} \left(f(x, y_1, \ldots, y_j + 1, \ldots, y_{n_i}) - f(x, y_1, \ldots, y_j - 1, \ldots, y_{n_i}) \right) \tag{3}
$$

for $j = 1, \ldots, n_i$, at neighbored grid points, see Exler et. al. [11] for the detailed procedure. If either $y_j + 1$ or $y_j - 1$ violates a bound, we apply a non-symmetric difference formula. Similarly, $d_y g_j(x, y)$ denote a difference formula for first derivatives at $g_j(x, y)$ computed at neighbored grid points.

The adaption of the continuous trust region radius must be modified, since a trust region radius smaller than one does not allow any further change of integer variables.

Therefore, two different radii are defined, Δ_k^c for the continuous and Δ_k^i for the integer variables. We prevent Δ_k^i from falling below one by $\Delta_{k+1}^i := \max(1, 2\Delta_k^i)$ and $\Delta_{k+1}^i := \max(1, 4\Delta_k^i)$. Note, however, that the algorithm allows a decrease of Δ_k^i below one. In this situation, integer variables are fixed and a new step is made only subject to the continuous variables. As soon as a new iterate is accepted, we set Δ_k^i to one in order to be able to continue optimization over the whole range of variables.

For solving continuous quadratic programming problems, we use the code QL of Schittkowski [29], which is based on an implementation of Powell [24]. The underlying primal-dual method of Goldfarb and Idnani [17] is particularly useful for designing a branch-and-bound algorithm for mixed-integer quadratic programs, e.g., by exploiting dual information for early branching. The corresponding code is called MIQL, see Lehmann et al. [20], and is used in all situations where we have to solve mixed-integer quadratic programs.

3 Numerical Results

We are interested in comparing several parameter settings to analyze their influence on the performance. We select 142 test problems out of 186 problems published in Schittkowski [32], which have not more than 20 integer or binary variables and not more than 400 continuous variables, to prevent extreme outliers. All test examples are provided with the best objective function value $f^\star$ we know, either obtained from analytical solutions, literature, or extensive numerical testing, see Schittkowski [32] for details. In seven cases, we found better solutions than those published in the literature.

Note again that our main paradigm is to proceed from non-relaxable integer variables and descent directions, which can be considered as very crude numerical approximations of partial derivatives by a difference formula. To be as close as possible to complex practical engineering applications, derivatives with respect to continuous variables are approximated by a forward difference formula. Integer derivatives are replaced by descent directions, see Exler et al. [11] for the detailed procedure. For binary variables or for variables at a bound, a forward or backward difference formula is applied, respectively.

First we need a criterion to decide whether the result of a test run can be considered as a successful return or not. Let $\epsilon_t > 0$ be a tolerance for defining the relative accuracy, and (x_k, y_k) the final iterate of a test run. If $f^\star = 0$, as in some of the academic test instances, we add the value one to the objective function. We call (x_k, y_k) a *successful* solution, if the relative error in the objective function is less than ϵ_t and if the maximum constraint violation is less than ϵ_t^2, i.e., if

$$f(x_k, y_k) - f^\star < \epsilon_t |f^\star| \tag{4}$$

and $\|g(x_k, y_k)^-\|_\infty < \epsilon_t^2$, where $g(x_k, y_k)^-$ represents the vector of constraint violations. We take into account that a code returns a solution with a better function value than the best known one, subject to the error tolerance of the allowed constraint violation.

The tolerance is smaller for measuring the constraint violation than for the error in the objective function, since we apply a relative measure in this case.

Optimization algorithms for continuous optimization have the advantage that in each step, local optimality criteria, the so-called KKT-conditions, can be checked. The algorithm is stopped as soon as an iterate is sufficiently feasible and satisfies these stationary conditions subject to a user-provided tolerance.

In mixed-integer optimization, however, we do not know how to distinguish between a *local* or *global* solution nor do we have any local criterion to find out whether we are sufficiently close to a solution or not. Thus, our algorithm stops if constraints are satisfied subject to a tolerance, and if no further progress is possible towards neighboring grid points as measured by a merit function. Thus, we distinguish between feasible, but non-global solutions, successful solutions as defined above, and false terminations. We call the return of a test run, say x_k and y_k, an *acceptable* solution, if the internal termination conditions are satisfied subject to a reasonably small tolerance $\epsilon = 10^{-6}$ and if instead of (4)

$$f(x_k, y_k) - f^\star \geq \epsilon_t |f^\star| \tag{5}$$

holds. For our numerical tests, we use $\epsilon_t = 0.01$. Note that our approach is based on a local search method, as in case of continuous optimization, and that global search is never started.

There is basic difficulty when evaluating statistical comparative scores by mean values for a series of test problems and different computer codes. It might happen that the less reliable codes do not solve the more complex test problems successfully, but the more advanced ones solve them with additional numerical efforts, say calculation time or number of function calls. A direct evaluation of mean values over successful test runs would thus penalize the more reliable algorithms.

A more realistic possibility is to compute mean values of the criteria we are interested in, and to compare the codes pairwise over sets of test examples, which are successfully solved by two codes. We then get a reciprocal $n_{code} \times n_{code}$ matrix, where n_{code} is the number of codes under consideration. The largest eigenvalue of this matrix is positive and we compute its normalized eigenvector from where we retrieve priority scores. In a final step, we normalize the eigenvectors so that the smallest coefficient gets the value one. The idea is known under the name *priority theory*, see Saaty [27] or the appendix, and has been used by Schittkowski [28] for comparing 27 optimization codes, see also Exler et al. [11] for another comparative study of mixed-integer codes.

We use the subsequent criteria to compare the robustness and efficiency of our codes:

n_{succ}	-	percentage of successful and acceptable test runs according to above definitions,
n_{acc}	-	percentage of acceptable, i.e., of non-global feasible solutions, see above definition,
n_{err}	-	percentage of errors, i.e., of terminations with IFAIL>0,
p_{func}	-	relative priority of equivalent function calls including function calls used for gradient approximations,
n_{func}	-	average number of equivalent function calls including function calls used for computing a descent direction or gradient approximations, evaluated over all successful test runs, where one function call consists of one evaluation of the objective function and all constraint functions at a given x and y,
p_{grad}	-	relative priority of number of iterations,
n_{grad}	-	average number of iterations or gradient evaluations subject to the continuous variables, respectively, evaluated over all successful test runs,
p_{time}	-	relative priority of execution times.
$time$	-	average execution times in seconds, evaluated over all successful test runs,

For the standard version of MISQP, partial derivatives subject to continuous variables are computed by forward differences. The following parameter settings are defined to compare several versions of MISQP against the standard options:

1	-	standard version
2	-	external function scaling
3	IOPT(1)=0	function scaling suppressed
4	LOPT(3)=F	internal scaling of matrix B_k suppressed
5	IDERIV=F	partial derivatives subject to integer variables computed by forward differences
6	IOPT(6)=0	restarts suppressed
7	IOPT(7)=1	L_∞ merit function
8	-	partial derivatives subject to continuous variables computed by two-sided differences
9	IOPT(2)=0	monotone trust region updates

We apply default parameter settings and tolerances unless changed as indicated in the table, with termination tolerance 10^{-6} of MISQP and 10^{-12} of MIQL, the mixed-integer quadratic programming code, see Lehmann et al. [20]. Maximum number of iterations is 500, the maximum number of iterations without improvements is 5, and the maximum number of branch-and-bound nodes for solving a mixed-integer quadratic programming subproblem is 1,000.

The Fortran codes are compiled by the Intel Visual Fortran Compiler 10.1 under Windows Vista and executed on an Intel Core(TM) i7-2720QM CPU with 2.20 GHz and 8 GB RAM.

code	$n_{succ}(\%)$	$n_{acc}(\%)$	$n_{err}(\%)$	p_{func}	n_{func}	p_{grad}	n_{grad}	p_{time}	*time*
1	88.0	9.9	2.1	1.8	529	1.6	20	2.3	0.19
2	88.0	10.6	1.4	1.8	586	1.6	21	2.4	0.18
3	84.5	9.2	6.3	1.8	475	1.6	19	2.4	0.16
4	62.0	35.2	2.8	1.5	377	1.4	13	1.8	0.12
5	83.1	12.7	4.2	1.8	667	2.5	34	2.5	0.22
6	64.1	32.4	3.5	1.0	273	1.0	10	1.0	0.05
7	88.0	11.3	1.4	2.1	696	2.0	27	2.9	0.24
8	89.4	8.5	2.1	2.5	839	1.6	21	2.1	0.18
9	83.1	14.8	2.1	1.6	487	1.5	18	2.4	0.16

Table 1: Performance Results for Different Parameter Settings of MISQP

To understand the fifth version, one has to take into account that although we proceed from non-relaxable model functions, our test problems nevertheless consist of analytical functions and are relaxable. In this case, we want to check whether analytical partial derivatives subject to integer variables, if exist, lead to better numerical results.

Note that one iteration corresponds to the solution of one mixed-integer quadratic programming problem. The number of equivalent is always relatively large, since $2n_c$ additional function evaluations are needed to approximate partial derivatives subject to continuous variables by a two-sided difference formula. To give an example, we need 360 additional function calls for each iteration to approximate derivatives for test problem PARALLEL.

We evaluate the performance of MISQP on a test set of academic test examples published in Schittkowski [32]. Each test problem comes with a function value which has been found in the literature or which has been obtained by extensive testing over several years. They are believed to represent the optimal solution, at least we did not find better results by our test runs. Most test problems are taken from the GAMS MINLPLib, see Bussieck, Drud, and Meeraus [5]. Table 1 shows numerical results obtained for the different parameter settings of the mixed-integer trust region method MISQP.

A couple of test runs were terminated by an error message, in most cases because of reaching maximum number of iterations or not being able to find a feasible point. In the first case, it is nevertheless possible that an optimal solution is obtained.

The most important conclusions are:

- The more sophisticated external function scaling procedure does not improve robustness or efficiency.
- Without proper scaling of the quasi-Newton matrix, the algorithm becomes much more unreliable.

- Analytical partial derivatives subject to the integer variables do not improve robustness or efficiency, and the number of iterations is enlarged.
- Internal restarts are extremely important despite higher number of function evaluations and calculation times. Otherwise, the code becomes unreliable.
- Change of the merit function does not influence performance significantly.
- More accurate partial derivatives subject to the continuous variables increase the number of function evaluations, but are a bit more reliable.
- Monotone trust region updates decrease the number of function calls, but less problems are successfully solved.

Finally, it might be interesting to compare performance results obtained for mixed-integer problems on the one hand and for the corresponding continuous relaxations on the other. Note that all of our codes are designed to handle integer variables as non-relaxable ones, even if continuous relaxations are available and thus also partial derivatives subject to integer variables. Nevertheless, our test problems are relaxable and MISQP can be applied for solving continuous optimization problems in an efficient way. The code is then a typical SQP-implementation with trust-region and a second-order stabilization.

Table 2 contains numerical results, where code number 10 stands for the relaxed version of MISQP and 11 for reference results obtained by the code NLPQLP of Schittkowski [30, 31]. Also in this case, we apply a one-sided difference formula for partial derivatives to avoid side effects due to inaccurate gradients.

The relaxed version of MISQP computes better optimal solutions for 106 test runs, as must be expected. Very surprisingly, the numerical solution of mixed-integer test problems requires significantly more function evaluations as the solution of the relaxed counterparts. We conclude that the relaxation of our test problems makes them more difficult to solve. Because of a large number of branch-and-bound steps when solving the mixed-integer quadratic subproblems, computation times for solving mixed-integer programs are much higher.

There is still another important conclusion. About 97 % of the relaxed mixed-integer test problems are successfully solved by MISQP in the sense that the code terminates at a feasible iterate where the optimality criteria are satisfied subject to a tolerance of 10^{-6}, which is either a local or a global solution. Thus, we should not expect that MISQP will be able to get significantly higher scores for n_{succ} for the much more complex mixed-integer version of our test problem suite. Another conclusion is that at least 16 % of the test problems are probably non-convex.

code	$n_{succ}(\%)$	$n_{acc}(\%)$	$n_{err}(\%)$	n_{func}	n_{grad}	*time*
1	88.0	9.9	2.1	529	20	0.19
10	81.0	15.8	3.5	837	26	0.06
11	83.8	4.2	12.0	370	19	0.01

Table 2: Performance Results of Mixed-Integer versus Continuously Relaxed Test Problems

4 Program Documentation

MISQP is implemented in Fortran, as close as necessary to F77, without global variables (COMMON) or 'tricky' Fortran constructs like EQUIVALENCE or ENTRY. Transfer to C by f2c is easily possible. Nonlinear functions and partial derivatives subject to continuous variables must be provided by reverse communication, the most flexible and most user-friendly procedure. Note that the total number of integer and boolean variables must be greater than zero.

MISQP is an implementation of the trust region SQP algorithm outlined in the previous section, in form of a Fortran subroutine. The mixed-integer quadratic programming problems are solved by the code MIQL of Lehmann et al. [20], an implementation of a branch-and-bound method. The underlying continuous quadratic programs are solved by an implementation of the primal-dual method of Goldfarb and Idnani [17] going back to Powell [24], see also Schittkowski [29]. Model functions and gradients are called by reverse communication. By default, first and second derivative approximations subject to integer variables are computed internally by forward and two-sided differences at neighbored grid points. The user has to provide functions and gradients in the same program, which executes MISQP according to the following rules:

1. Choose starting values for the optimization variables, and store them in X, first the continuous, then the binary followed by the integer variables. Initialize ROPT, IOPT, and LOPT, at least with default values (0.0, -1, TRUE).

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

3. Compute partial derivatives of objective function and constraints subject to the continuous variables 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 required, also partial derivatives subject to the integer variables specified in IDERIV can be provided.

4. Set IFAIL=0 and execute MISQP.

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

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

7. If MISQP returns with IFAIL=-1, compute objective and constraint function values at the parameter values found in X, store them in F and G, and call MISQP again.

8. If MISQP terminates with IFAIL=-2, compute gradient values subject to actual variable stored in X, and store them in DF and DG. Only partial derivatives subject to the continuous variables and the integer variables specified in IDERIV need to be provided. Then call MISQP again.

Note that derivatives subject to integer variables are approximated by MISQP internally using a difference formula, if not prided by the user through IDERIV. The calling sequence and the meaning of the parameters of MISQP 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 the output and other related software documentations.

Usage:

```
      CALL  MISQP(     M,      ME,   MMAX,     N,    NBIN,
     /              NINT,       X,      F,     G,      DF,
     /                DG,      XL,     XU,   ACC,   MAXIT,
     /             MNFS,  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, binary and integer 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. Moreover, NBIN+NINT must be greater than zero.
X(N):	Input and output vector containing 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. It is sufficient to determine the gradients subject to the continuous variables and variables specified in IDERIV. See also description of 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 must be equal to MMAX. It is sufficient to determine the gradients subject to the continuous variables and variables specified in IDERIV. See also description of IDERIV below.
XL(N),XU(N) :	On input, the one-dimensional arrays XL and XU must contain the upper and lower bounds of the variables, first for the continuous, then for the binary and subsequently for the integer variables.
ACC:	Input parameter defining the tolerance for termination.
MAXIT :	Maximum number of iterations.
MNFS :	Maximum number of feasible steps without improvements, where the relative change of objective function values is measured by ACC and feasibility is measured by sqrt(ACC). Must be greater than 1.
MAXNDE :	Maximum number of branch-and-bound nodes for solving the mixed-integer quadratic programming subproblem (1000).
IPRINT :	Specification of the desired output level. 0 : No output of the program. 1 : Only a final convergence analysis is given.

2 : One line of intermediate results is printed in each iteration.

3 : More detailed information is printed for each iteration.

IOUT : Positive integer number indicating the desired output unit number, i.e., all write-statements start with 'WRITE(IOUT,... '.

IFAIL : The parameter shows the reason for terminating a solution process. Initially IFAIL must be set to zero. On return IFAIL could contain the following values:

-2 : Compute new gradient values for continuous variables and variables specified in IDERIV in DF and DG. See description of DF, DG and IDERIV.

-1 : Compute new function values in F and G, see above.

0 : Optimality conditions satisfied.

1 : Termination after MAXIT iterations.

2 : Trust region radius lower than ACCQP.

3 : Penalty parameter SIGMA tends to infinity.

4 : Termination at infeasible iterate.

5 : Termination with zero trust region for integer variables.

6 : Length of a working array is too short.

7 : False dimensions, e.g., M>MMAX.

8 : Lower or upper bound of a variable violates initial value.

9 : Linear constraints are inconsistent.

11: The continuous quadratic solver QL could not solve a quadratic program after a maximal number of (40*(N+M)) iterations.

12 : The termination accuracy is insufficient for the continuous quadratic solver QL to satisfy the convergence criterion.

13 : The continuous quadratic solver QL terminated due to an internal inconsistency, division by zero.

14 : Numerical instabilities prevent successful termination of continuous quadratic solver QL.

15 : Termination due to more than MNFS steps without improvements of feasible function values.

>90 : QP solver terminated with an error message IFQL and IFAIL is set to IFQL + 100.

IDERIV(NBIN+NINT) : Logical input array specifying integer variables whose derivatives are provided by the user and stored in DG and DF.

IDERIV(I)=TRUE : Column NCONT+I of DG and position

NCONT+I of DF are set by the user.

IDERIV(I)=FALSE : Column NCONT+I of DG and position NCONT+I of DF are set by MISQP by evaluating functions at neighbored grid points.

Here we have NCONT = N - NINT - NBIN.

ROPT(60) : Double precision option array, to be initialized with -1.0D0 for default parameter setting. Note that ROPT(41) to ROPT(60) are passed to the MIQP solver and that our implementation MIQL may be replaced by another one.

- ROPT(1) Termination tolerance of the QP solver for several tests, e.g., whether optimality conditions are satisfied or whether a number is considered as zero or not (ACCQP, 1.0E-12).
- ROPT(2) Factor for increasing a penalty parameter, must be greater than one (RPEN, 10).
- ROPT(3) Factor for increasing the internal descent parameter DELTA, see ROPT(5). ROPT(3) must be less than one (DLTDEC, 0.1).
- ROPT(4) Initial penalty parameter, greater than one (SIGMA, 1000).
- ROPT(5) Initial scaling parameter, smaller than one (DELTA, 0.05).
- ROPT(6) Initial continuous trust region radius, greater than zero (TRUSTC, 10).
- ROPT(7) Initial integer trust region radius, not less than one (TRUSTI, 10).
- ROPT(8) Scaling bound in case of IOPT(1)>0, i.e., function values are scaled only if their absolute initial values are greater than the bound (SCBND, 1.0).

IOPT(60) : Integer option array, to be initialized with -1 for default parameter setting. Note that IOPT(41) to IOPT(60) are passed to the MIQP solver and that our implementation MIQL may be replaced by another one.

- IOPT(1) Internal scaling of function values (SCALE, 1).

  0 : No scaling.

  1 : Scaling subject to absolute initial function values, if greater than one or ROPT(8), respectively.
- IOPT(2) Maximum number of successive iterations which are considered for the non-monotone trust region algorithm, must be less than 100 (NONMON, 10).

IOPT(3) Print level of the subproblem solver MIQL (IPRQP, 2).

IOPT(4) Output for number of gradient evaluations.

IOPT(5) Output for number of function evaluations.

IOPT(6) Maximum number of successive restarts without improving solution. Setting might lead to better results, but increases the number of function evaluations (MRS, 2).

IOPT(41) Branching rule (IBR, 1):

1 : Maximal fractional branching.

2 : Minimal fractional branching.

IOPT(42) Node selection strategy (INS, 3):

1 : Best of all (large search trees).

2 : Best of two (warmstarts, less memory for search tree).

3 : Depth first (good warmstarts, less memory, many QPs solved).

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

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

LOPT(60) : Logical option array, to be initialized with TRUE for default parameter setting. Note that LOPT(41) to LOPT(60) are passed to the MIQP solver and that our implementation MIQL may be replaced by another one.

LOPT(2) Internal scaling of continuous variables (SCALE, TRUE).

LOPT(3) Modification of Hessian approximation to get more accurate search directions. Calculation time is increased in case of a large number of integer variables (BMOD, TRUE).

RW(LRW) : Real working array of length LRW.

LRW: Input parameter defining the length of RW, must be at least
7*N*N/2 + MMAX0*N + 105*N + 37*MMAX0 + 3*MAXNDE
+ 3*M*M/2 + 4*M*N + 500, where
MMAX0 = M + ME + NINT + NBIN + 20.

IW(LIW) : Integer working array of length LIW.

LIW: Input parameter defining the length of IW, must be at least
14*N + 6*MMAX0 + 6*MAXNDE + 150.

LW(LLW) : Logical working array of length LLW.

LLW: Input parameter defining the length of LW, must be at least

4*N + MMAX0 + 150.

MISQP must be linked with the calling routine of the user and the mixed-integer quadratic optimization code MIQL.

We cannot guarantee convergence to a global optimal solution, even not in the convex case. It is possible that the algorithms stops with IFAIL=0 although the global optimum subject to the integer variables has not been approached. A typical situation is successive reduction of trust regions of the integer variables until falling beyond one, where no further progress can be obtained.

In case of successful termination, the code passes the best computed solution to the calling program together with corresponding function values. The derivative arrays DF and DG do not always contain valid partial derivatives when leaving MISQP.

5 Example

To give an example how to organize the code, we consider a pseudo-convex test problem of Westerlund and Pörn [35], see also problem WP02 in Schittkowski [32], with one continuous and one integer variable,

$$x \in \mathbb{R}, y \in \mathbb{N}: \quad \begin{aligned} &\min \frac{(x-3)^2 - 10x}{3x + y + 1} \\ &5y - (x-7)^2 \geq 0 \\ &1.8y - x \geq 0 \\ &1 \leq x \leq 8 \\ &1 \leq y \leq 8 \end{aligned} \tag{6}$$

The Fortran source code for executing MISQP is listed below.

```
      IMPLICIT          NONE
      INTEGER           NMAX, MMAX, MMAX0, MAXNDE, LRW, LIW, LLW
      PARAMETER        (NMAX   = 2,
     /                  MMAX   = 2,
     /                  MAXNDE = 1000,
     /                  MMAX0  = 2*MMAX + NMAX + 20,
     /                  LRW    = 7*NMAX*NMAX/2 + MMAX0*NMAX + 105*NMAX
     /                           + 37*MMAX0 + 3*MAXNDE + 3*MMAX*MMAX/2
     /                           + 4*MMAX*NMAX + 500,
     /                  LIW    = 14*NMAX + 6*MMAX0 + 6*MAXNDE + 150,
     /                  LLW    = 4*NMAX + MMAX0 + 150)
      INTEGER           IW(LIW), N, NCONT, NINT, NBIN, M, ME, IOUT,
     /                  MAXIT, MNFS, IPRINT, IFAIL, I, J, IOPT(60)
      DOUBLE PRECISION  RW(LRW), F, X(NMAX), G(MMAX), DF(NMAX),
     /                  DG(MMAX,NMAX), XL(NMAX), XU(NMAX), ACC, EPS,
     /                  EPSREL, FBCK, GBCK(MMAX), ROPT(60)
      LOGICAL           LW(LLW), IDERIV(NMAX), LOPT(60)
C
C   Set some constants and initial values
C
```

```
      N     = 2
      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 MISQP
C
      ACC       = 1.0D-6
      MAXIT     = 100
      MNFS      = MAXIT
      EPS       = 1.0D-7
      IPRINT    = 2
      IOUT      = 6
      IDERIV(1) = .FALSE.
C
C   Optimization block (reverse communication)
C
      IFAIL = 0
      I     = 0
    1 CONTINUE
C============================================================
C   This is the main block to compute all function values
C   simultaneously. The block is executed either for computing
C   a search direction or for approximating gradients by forward
C   differences.
C
      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)
C
C============================================================
      IF (IFAIL.EQ.-1) GOTO 4
      IF (I.GT.0) GOTO 3
    2 CONTINUE
      FBCK = F
      DO J=1,M
         GBCK(J) = G(J)
      END DO
      I = 0
    5 I = I + 1
      EPSREL = EPS*DMAX1(1.0D0,DABS(X(I)))
      X(I) = X(I) + EPSREL
      GOTO 1
    3 CONTINUE
      DF(I) = (F - FBCK)/EPSREL
      DO J=1,M
         DG(J,I) = (G(J) - GBCK(J))/EPSREL
      END DO
      X(I) = X(I) - EPSREL
```

```
      IF (I.LT.NCONT) GOTO 5
      F = FBCK
      DO J=1,M
         G(J) = GBCK(J)
      END DO
    4 CONTINUE
C
      CALL MISQP(     M,     ME,      M,      N,   NBIN,
     /             NINT,      X,      F,      G,     DF,
     /               DG,     XL,     XU,    ACC,  MAXIT,
     /             MNFS, MAXNDE, IPRINT,   IOUT,  IFAIL,
     /           IDERIV,   ROPT,   IOPT,   LOPT,     RW,
     /              LRW,     IW,    LIW,     LW,    LLW )
      IF (IFAIL.EQ.-1) GOTO 1
      IF (IFAIL.EQ.-2) GOTO 2
C
C   End of main program
C
      STOP
      END
```

The following output should appear on screen:

```
 ------------------------------------------------------------------------
           Start of the Mixed-Integer SQP Algorithm MISQP
                      Version 6.4 (Nov 2012)
 ------------------------------------------------------------------------

 Parameters:

   Number of all variables:                        2   N
   Number of continuous variables:                 1   NCONT
   Number of binary variables:                     0   NBIN
   Number of integer variables:                    1   NINT
   Number of all constraints:                      2   M
   Number of equality constraints:                 0   ME
   Termination accuracy:                  0.100D-05   ACC
   Maximum number of iterations:                 100   MAXIT
   Number of steps without progress:             100   MNFS
   Maximum number of nodes:                     1000   MAXNDE
   Output level:                                   2   IPRINT

 Non-default options:

   Initial integer trust region radius:    0.700D+01   TRUSTI,ROPT( 7)

 Output in the following order:

   IT    - iteration number
   F     - objective function value
   MCV   - maximum constraint violation
   SIGMA - penalty parameter
   IL    - number inner loops
   DMAXC - maximum norm of continuous step D_C
   D1B   - 1-norm of binary step DELTA_B
   DMAXI - maximum norm of integer step D_I

 IT       F         MCV      SIGMA    IL   DMAXC      D1B      DMAXI
------------------------------------------------------------------------
  1 -0.10000D+01  0.10D+01  0.10D+04   1  0.31D+01  0.00D+00  0.10D+01
```

```
 2 -0.21093D+01  0.15D+00  0.10D+04   1  0.85D+00  0.00D+00  0.10D+01
 3 -0.20194D+01  0.00D+00  0.10D+04   1  0.58D-01  0.00D+00  0.00D+00
 4 -0.20220D+01  0.00D+00  0.10D+04   1  0.27D+00  0.00D+00  0.00D+00
 5 -0.20310D+01  0.00D+00  0.10D+04   1  0.42D+00  0.00D+00  0.00D+00
 6 -0.20369D+01  0.00D+00  0.10D+04   1  0.60D-01  0.00D+00  0.00D+00
 7 -0.20370D+01  0.00D+00  0.10D+04   1  0.63D+00  0.00D+00  0.10D+01
 8 -0.21647D+01  0.29D+00  0.10D+04   1  0.64D+00  0.00D+00  0.10D+01
 9 -0.20370D+01  0.00D+00  0.10D+04   1  0.87D+00  0.00D+00  0.10D+01

--- FINAL CONVERGENCE ANALYSIS ---

  Objective function value:        F(X)  = -0.20370364D+01
  Approximation of solution:       X     =
       0.43368425D+01  0.30000000D+01
  Constraint function values:      G(X)  =
       0.25508361D+00  0.10631575D+01
  Distances from lower bounds:     XL-X  =
      -0.33368425D+01 -0.20000000D+01
  Distances from upper bounds:     XU-X  =
       0.36631575D+01  0.50000000D+01
  Number of function calls:        NFUNC =       26
   - within TR method:             NF_TR =        9
   - integer derivatives:          NF_2D =       17
  Number of gradient calls:        NGRAD =        9
  Number of calls of QP solver:    NQL   =       12
   - 2nd order corrections:        NQL2  =        0
  Number of B&B nodes:             NODES =       30
  Termination reason:              IFAIL =        0

--- UNSCALED VALUES ---

  Objective function value:        F(X)  = -0.24444437D+01
  Constraint function values:      G(X)  =
       0.79075919D+01  0.10631575D+01
```

References

[1] Audet C., Dennis J.E. (2001): *Pattern search algorithm for mixed variable programming*, SIAM Journal on Optimization, Vol. 11, 573–594

[2] Borchers B., Mitchell J.E. (1994): *An improved branch and bound algorithm for mixed integer nonlnear programming,* Computers and Operations Research, Vol. 21, No. 4, 359-367

[3] Bünner M.J., Schittkowski K., van de Braak G. (2004): *Optimal design of electronic components by mixed-integer nonlinear programming*, Optimization and Engineering, Vol. 5, 271-294

[4] Burke J.V. (1992): *A robust trust region method for constrained nonlinear programming problems*, SIAM Journal on Optimization, Vol. 2, 325–347

[5] Bussieck M.R., Drud A.S., Meeraus A. (2007): *MINLPLib - A collection of test models for mixed-integer nonlinear programming*, GAMS Development Corp., Washington D.C., USA

[6] Byrd R., Schnabel R.B., Schultz G.A. (1987): *A trust region algorithm for nonlinearly constrained optimization*, SIAM Journal of Numerical Analysis, Vol. 24, 1152–1170

[7] Celis M.R. (1983): *A trust region strategy for nonlinear equality constrained optimization*, Ph.D. Thesis, Department of Mathematics, Rice University, USA

[8] Conn A.R., Gould I.M., Toint P.L. (2000): *Trust-Region Methods*, SIAM, Philadelphia

[9] Duran M., Grossmann I.E. (1986): *An outer-approximation Algorithm for a class of mixed-integer nonlinear programs*, Mathematical Programming, Vol. 36, 307–339

[10] Exler O., Schittkowski K. (2007): *A trust region SQP algorithm for mixed-integer nonlinear programming*, Optimization Letters, Vol. 1, 269-280

[11] Exler O., Lehmann T., Schittkowski K. (2010): *A comparative study of SQP-type algorithms for nonlinear and nonconvex mixed-integer optimization*, submitted for publication

[12] Fletcher R. (1981): *Practical Methods of Optimization. Volume 2, Constrained Optimization*, Wiley, Chichester

[13] Fletcher R. (1982): *Second order correction for nondifferentiable optimization*, in: Watson G.A. (Hrsg.): *Numerical analysis*, Springer Verlag, Berlin, 85–114

[14] Fletcher R., Leyffer S. (1994): *Solving mixed integer nonlinear programs by outer approximation*, Mathematical Programming, Vol. 66, 327–349

[15] Floudas C.A. (1995): *Nonlinear and Mixed-Integer Optimization*, Oxford University Press, New York, Oxford

[16] Gill P.E., Murray W., Wright M. (1981): *Practical Optimization*, Academic Press, New York

[17] Goldfarb D., Idnani A. (1983): *A numerically stable method for solving strictly convex quadratic programs*, Mathematical Programming, Vol. 27, 1–33

[18] Grossmann I.E., Kravanja Z. (1997): *Mixed-integer nonlinear programming: A survey of algorithms and applications*, in: Conn A.R., Biegler L.T., Coleman T.F., Santosa F.N. (eds.): *Large-Scale Optimization with Applications, Part II: Optimal Design and Control*, Springer, New York, Berlin

[19] Gupta O. K., Ravindran V. (1985): *Branch-and-bound experiments in convex nonlinear integer programming*, Management Science, Vol. 31, 1533–1546

[20] Lehmann T., Schittkowski K., Spickenreuther T. (2009): *MIQL : A Fortran code for convex mixed integer quadratic programming by branch-and-bound - User's guide*, Report, Department of Computer Science, University of Bayreuth, Germany

[21] Leyffer S. (2001): *Integrating SQP and branch-and-bound for mixed integer nonlinear programming*, Computational Optimization and Applications, Vol. 18, 295–309

[22] Li H.-L., Chou C.-T. (1994): *A global approach for nonlinear mixed discrete programming in design optimization*, Engineering Optimization, Vol. 22, 109–122

[23] Moré J.J. (1983) *Recent developments in algorithms and software for trust region methods*, in: Bachem A., Grötschel M., Korte B. (eds.): *Mathematical Programming: The State of the Art*, Springer, Berlin, 258–287

[24] Powell M.J.D. (1983): *On the quadratic programming algorithm of Goldfarb and Idnani*. Report DAMTP 1983/Na 19, University of Cambridge, Cambridge

[25] Powell M.J.D. (1984): *On the global convergence of trust region algorithms for unconstrained minimization*, Mathematical Programming, Vol. 29, 297–303

[26] Powell M.J.D., Yuan Y. (1991): *A trust region algorithm for equality constrained optimization*, Mathematical Programming, Vol. 49, 189–211

[27] Saaty T.L. (1977): *A scaling method for priorities in hierarchical structures*, Journal of Mathematical Psychology, Vol. 15, 234-281

[28] Schittkowski K. (1980): *Nonlinear Programming Codes - Information, Tests, Performance*, Lecture Notes in Economics and Mathematical Systems, Vol. 183, Springer

[29] Schittkowski K. (2003): *QL: A Fortran code for convex quadratic programming - user's guide, version 2.11*, Report, Department of Mathematics, University of Bayreuth

[30] Schittkowski K. (1985): *NLPQL: A FORTRAN subroutine solving constrained nonlinear programming problems*, Annals of Operations Research, Vol. 5, 485-500

[31] Schittkowski K. (2006): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search - user's guide*, Report, Department of Computer Science, University of Bayreuth

[32] Schittkowski K. (2012): *A collection of 186 test problems for nonlinear mixed-integer programming in Fortran - User's Guide*, Report, Department of Computer Science, University of Bayreuth, Germany

[33] Stoer J. (1985): *Foundations of recursive quadratic programming methods for solving nonlinear programs,* in: K. Schittkowski (ed.): *Computational Mathematical Programming*, NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 15, Springer

[34] Toint P.L. (1997): *A nonmonotone trust-region algorithm for nonlinear optimization subject to convex constraints*, Mathematical Programming, Vol. 77, 69–94

[35] Westerlund, Pörn (2002): *Solving pseudo-convex mixed integer optimization problems by cutting plane techniques*, Optimization and Engineering, Vol. 3, 253-280

[36] Yuan Y.-X. (1985): *On the superlinear convergence of a trust region algorithm for nonsmooth optimization*, Mathematical Programming, Vol. 31, 269–285

[37] Yuan Y.-X. (1995): *On the convergence of a new trust region algorithm*, Numerische Mathematik, Vol. 70, 515–539