Alpha-ECP, Version 5.101
An Interactive MINLP-Solver Based on the Extended Cutting Plane Method

Tapio Westerlund and Kurt Lundqvist

Process Design Laboratory, Abo Akademi University
Biskopsgatan 8, FIN-20500 ABO, Finland

ABSTRACT
The present user’s guide is given for the upgraded version 5.101 of the Alpha-ECP solver. Alpha-ECP is a MINLP (Mixed-Integer Non-Linear Programming) solver based on the extended cutting plane method. The solver can be applied to general MINLP problems and global optimal solutions can be ensured for problems with a pseudo-convex objective function and pseudo-convex inequality constraints. A brief documentation of the program is given and some example problems are provided to illustrate the numerical procedure.

KEYWORDS
INTRODUCTION


There are a great number of programs available for LP and NLP problems but only a very few interactive programs available for the solution of MINLP problems. Among these few programs able to solve MINLP problems we find the Outer Approximation method, by Viswanathan and Grossmann (1990) implemented in the GAMS-DICOPT++ solver. An SQP based Branch and Bound method by Leyffer (2001) has been implemented in a solver called MINLP-BB. For some special classes of global optimization problems the GOP program by Visweswaran and Floudas (1996) and the BARON Global Optimization Software by Sahinidis (1996) are available.


The ECP method is an extension of Kelley's cutting plane (CP) method which was originally given for convex NLP problems (Kelley, 1960). In Westerlund and Pettersson (1995) the method was extended to convex MINLP problems and in Westerlund et al. (1998) further extended to MINLP problems with pseudo-convex constraints. An analysis of the global convergence properties of the method for MINLP and NLP problems with quasi-convex constraints was given in Still and Westerlund (1997, 2000). The method was further extended in Westerlund and Pörn (1999), Pörn and Westerlund (2000) and Westerlund and Pörn (2002) and the actual version of the method converges to the global optimal solution for non-convex MINLP problems having a pseudo-convex objective function and pseudo-convex inequality constraints.

The method requires only the solution of a MILP sub problem in each iteration. The MILP sub problems may be solved to optimality, but can also be solved to feasibility or only to an integer relaxed solution in intermediate iterations. This makes the ECP algorithm efficient and easy to implement. In the present implementation we have used a commercial MILP solver, CPLEX, from CPLEX Inc. to solve the MILP sub problems. The main ECP procedures are written in FORTRAN-90 and the program interface in Visual Basic for Windows NT, 2000 or XP. The problems to be solved can be supplied as text files in an extended LP format or if the constraints cannot be given in explicit form, then they can be given as user supplied sub programs. The problem files written in the extended LP format are automatically converted to FORTRAN-90 sub programs and linked as DLL files together with the ECP-solver. Another C implementation of the ECP method on UNIX platforms is given in Skrifvars H. (1998).
FORMULATION OF THE MINLP PROBLEM

The following is a description of the MINLP problem which can be solved to global optimum with the solver. The MINLP problem corresponding to the method may be formulated as follows,

$$\min_{z \in N \cap L} f(z)$$

$$N = \{z \mid g(z) \leq 0\}$$

$$L = \{z \mid Az \leq a, Bz = b\} \cap X \times Y$$

We will call this problem (P). The vector $z$, of variables, consists of a continuous part $x$ and an integer part $y$ i.e. $z^T = (x^T, y^T)$. The set $N$ is defined by a finite number of once differentiable functions $g$ defined on the linear set $L$ which is defined by all linear constraints in the problem and bounds on the variables. $A$ and $B$ are matrices and $a$ and $b$ vectors of appropriate dimensions. $X$ and $Y$ are hyper-rectangles that contain suitable bounds for the continuous and integer variables respectively. The feasible region $N \cap L$ of this problem is assumed to be non-empty, i.e. there exists a point $z \in N \cap L$ that satisfies all constraints and integer restrictions. If every function $g$ is pseudo-convex then the feasible region of the continuous relaxation of (P) is a convex set. If also $f$ is pseudo-convex in $L$ the proposed method converges to the global optimal solution of the problem (P).

The actual implementation of the ECP method, based on the extensions in Westerlund and Pörn (2002)), ensures the global optimality of MINLP problems having a pseudo-convex objective function, linear equality and inequality constraints as well as pseudo-convex inequality constraints.

At this stage one clarification may be mentioned. A problem with a non-linear objective function, $f$, can always be rewritten to a problem with a linear objective function by introducing an additional variable, $\mu$, and an additional constraint

$$f(z) - \mu \leq 0$$

The new problem will then be to minimize the additional variable, $\mu$, subject to the original constraints and the new constraint (1). Note, however, that if the method requires the constraints to be pseudo-convex, in order to ensure convergence to the global optimum, then the problem cannot be rewritten by using the constraint (1) only. Although the sum of two convex functions result in a convex function, a linear function added to or subtracted from a pseudo-convex function need not generally result in a pseudo-convex function (Westerlund and Pörn (2002)). The LHS of (1) is not generally pseudo-convex although, $f$, is pseudo-convex.

In Westerlund and Pörn (1999), Pörn and Westerlund (2000) and Westerlund and Pörn (2002) it was, however, found that a pseudo-convex objective function can be handled in a slightly different way, resulting in an algorithm by which a problem (P) where the objective function is pseudo-convex can be solved to global optimum. In this modification of the method, a sequence of MINLP problems with linear objective function and pseudo-convex constraints are solved. The ECP method given in Westerlund et al. (1998) and Still and Westerlund (1997,
2000) can, thus, be used to solve the sub problems in the new algorithm. Therefore, before we proceed to the modified ECP method for the problem \((P)\) let us first consider the solution strategy for a problem \((P)\) with a linear objective function.

The ECP method in Westerlund et al. (1998) and Still and Westerlund (1997, 2000), as implemented in the version 3.0 of the solver, converges to the global optimal solution if \(f\) is a linear function, i.e. \(f(z) = c^T z\). The case with a convex objective function is also included in that strategy. This is because after the introduction of the dummy variable, \(\mu\), the problem is equivalent to minimizing \(\mu\) subject to the original constraints plus the additional convex constraint \(f(z) - \mu \leq 0\). The rewriting procedure is, of course, also valid if \(f\) is pseudo-convex but, as mentioned above, in the latter case the additional constraint will no longer, generally, be pseudo-convex and the global convergence properties of the method as in Westerlund et al. (1998) are thus lost. The generalizations in Westerlund and Pörn (1999, 2002) and Pörn and Westerlund (2000), were therefore considered in order to cover the mentioned situation rigorously.

**THE ECP METHOD FOR A LINEAR OBJECTIVE FUNCTION**

The problem with a linear objective and pseudo-convex inequality constraints is solved by the \(\alpha\)ECP method (Westerlund et al. (1998)) by generating a sequence of points converging to the global optimal solution of the problem \((P)\). This is done by solving a sequence of MILP sub-problems. The algorithm adds more and more linear constraints, \(l_j(z)\), to a MILP problem originally consisting only of the linear constraints. The MILP sub-problems are of the form

\[
\begin{align*}
\min_{z \in \Omega_k} c^T z \\
\Omega_k = L \cap \{ z \mid l_j(z) \leq 0, j = 1,2,\ldots,J_k \} \\
k = 0,1,2,\ldots,K
\end{align*}
\]

The sub-problem solved in iteration \(k\) is called \((P^k)\) and include \(J_k\) additional linear constraints. The linear constraints, \(l_j(z)\), in \((P^k)\) are either linearizations corresponding to particular non-linear constraints obtained at previous solution points or redefined old linearizations. The procedure is initialized with \(\Omega_0 = L\). The linear functions added in iteration \(k\) are of the form

\[
l_j(z) = \frac{\tilde{g}_j}{\alpha_j} + \nabla \tilde{g}_j^T (z - z_k)
\]

\[
j \in \{ J_{k-1} + 1, \ldots, J_k \}
\]

Each \(\tilde{g}_j\) corresponds to the function value of a particular non-linear constraint \(g_j(z), i = 1,\ldots,m\), with positive value at \(z_k\). The vector \(\nabla \tilde{g}_j\) is the corresponding gradient of the non-linear constraint at \(z_k\). If only one linear constraint is added in each iteration then \(\tilde{g}_j = \max\{g_j(z_k)\}\) and \(J_k = J_{k-1} + 1\). \(\alpha_j^k\) is a scalar parameter. Initially \(\alpha_j^0 = 1\), but these values are sequentially increased in order to make all linearizations feasible under-estimators (or valid cutting planes) at termination. Clearly \(\alpha_j^k = 1\) is enough for all convex constraints.
The $\alpha^k_j$ parameter may be modified by either of the following expressions,

$$\alpha^k_j = \beta \cdot \alpha^{k-1}_j$$

(4)

$$\alpha^k_j = \gamma \cdot \alpha^{k-1}_j$$

(5)

$\beta$ and $\gamma$ are positive scalars, greater than 1.

In order to ensure convergence to the global optimal solution three essential steps are required:

I. If $z_k$ is infeasible in $(P)$ then this point is excluded from all subsequent iterations by each of the constraints $l_j(z) \leq 0$ added at the point $z_k$ (since $l_j(z_k) = \tilde{g}_j > 0$).

II. Every $l_j(z)$ added to the sub-problems must, at termination, be a valid cutting plane (not cutting off parts of $N \cap L$). In order to ensure this an up-dating strategy for the $\alpha^k_j$-parameters is required. The up-dating strategy, Eqs. (4-5), is activated before termination and after detection of infeasible MILP sub-problems.

III. A solution $z_k$ to $(P^k)$ is optimal in $(P)$ if $z_k$ is optimal in $(P^k)$, $z_k \in N \cap L$ and $N \cap L \subseteq \Omega$. That is $z_k$ is optimal in $(P)$ if it is optimal in $(P^i)$, feasible in $(P)$ and all $l_j(z)$ are valid cutting planes.

The first termination criteria, i.e. the test for feasibility is given by,

$$\max_j \{g_j(z_k)\} \leq \varepsilon_z$$

(6)

and the second termination criteria, i.e. the test that the linear constraints are valid cutting planes, not cutting off any part of the feasible region, $N \cap L$, is given by

$$\alpha^k_j \geq \frac{\tilde{g}_j}{h_j}, \quad j = 1,2,\ldots, J_k$$

(7.a)

where $h_j = \tilde{g}_j$ for each cutting plane that corresponds to a convex constraint and $h_j = \varepsilon_h$ if the cutting plane corresponds to a pseudo-convex constraint. Both $\varepsilon_z$ and $\varepsilon_h$ are small positive tolerances, depending on the problem. A detailed description and a proof of convergence of the method is given in Still et al. (2000) and Westerlund et al. (1998). In Westerlund and Pörn (2002) an important additional observation was made. If cutting planes are generated such that they cannot cut off any point or region on a larger distance than $\varepsilon_z$ from any of the level-curves $\tilde{g}(z) = \varepsilon_z$, for pseudo-convex functions, then the $h_j$ value in the denominator of (7.a) is, given by,

$$h_j = \varepsilon_z \cdot \sqrt{\nabla \tilde{g}_j^T \nabla \tilde{g}_j}$$

(7.b)

where $\sqrt{\nabla \tilde{g}_j^T \nabla \tilde{g}_j}$ is the norm of the gradient calculated at the point where the corresponding cutting plane was generated.
INCORPORATION OF A PSEUDO-CONVEX OBJECTIVE FUNCTION

Now we consider problem \((P)\) equipped with a pseudo-convex objective function. As a first approach we will construct a procedure that computes a sequence of improving upper bounds on the objective. The limit point of this sequence can be proven to be the optimum of \((P)\). This procedure can be viewed as a sequence of MINLP sub-problems \((P_\mu^n)\) of the form:

\[
\min_{z: N_\mu \cap L_\mu} \mu \\
N_\mu = \left\{ z \in N \mid f(z) \leq f(z^*_r) \right\} \\
L_\mu = \left\{ z \in L \mid \mu \leq f(z^*_r); f(z^*_r) + \nabla f(z^*_r)^T (z - z^*_r) - \mu \leq 0; \ l = 1,2,\ldots,n \right\}
\]

The set \(N_\mu\) contains all non-linear constraints from \(N\) plus an additional, so-called, reduction constraint \(f(z) \leq f(z^*_r)\). The linear set \(L_\mu\) contains all original linear constraints plus \(n\) linearizations of the objective function constraint (1) at the active contour. The index \(r\) counts the number of different contours visited and \(l\) is an index counting the solutions at the \(r\):th contour. The procedure is initialized with \(N_0 = N\) and \(L_0 = L\). The solution to this “feasibility” problem is a point \(z^*_1 \in N \cap L\) and the first proper upper bound \(f(z^*_1)\) is found. The pseudo-convex objective in \((P)\) is then, in the first real iteration, replaced with a linearization and the pseudo-convex reduction constraint \(f(z) \leq f(z^*_r)\) is included. If the reduction constraint is active at the next solution a new linearization is added to \(L_1\) and \(L_2\) will contain two linearizations. On the other hand, if the reduction constraint is redundant, a strict improvement in \(f(z)\) is obtained and all previous linearizations of \(f(z)\) will be replaced by a single linearization at the solution. Thus, \(L_1\) will contain one linearization and the reduction constraint is strengthened to \(f(z) \leq f(z^*_1)\). Note that problem \((P_\mu^n)\) has a linear objective and pseudo-convex inequality constraints and can therefore be solved to global optimality by the \(\alpha\)ECP method in Westerlund et al. (1998). Since the procedure is iterative it can be view at an MILP-level instead of at the MINLP-level. Thus, the solution procedure can fully be integrated within the framework of the \(\alpha\)ECP approach. This means that we can also use previously generated cutting planes for the reduction constraint and the original constraints when solving subsequent sub-problems. Also note that every sub-problem does not need to be solved to optimality, just to feasibility, i.e. to satisfy the reduction constraint. At this point an improved upper bound is found. Only the final MINLP problem \((P_\mu^n)\) must be solved to optimality. Other computational improvements are discussed below.

PROOF OF CONVERGENCE

The proof is given in Pörn and Westerlund (2000) as well as in Westerlund and Pörn (2002). The proof relies mainly on the definition of a pseudo-convex function. A function \(f\) is pseudo-convex on a convex set \(S \subset R^n\) if \(f(z_1) < f(z_2) \Rightarrow \nabla f(z_2)^T (z_1 - z_2) < 0\) for every \(z_1, z_2 \in S\).

**Theorem of optimality.** If \(z^*\) is a solution to \((P_\mu^n)\) with \(\mu = f(z^*_r)\) then \(z^*\) is optimal in \((P)\).

**Proof.** Since we are minimizing \(\mu\) at least one of the constraints \(f(z^*_r) + \nabla f(z^*_r)^T (z - z^*_r) - \mu \leq 0\) must be active at the solution \(z^*\). Assume that constraint \(l = 1\) is active. This implies that \(f(z^*_r) + \nabla f(z^*_r)^T (z^* - z^*_r) - \mu = 0\). From the definition of pseudo-
convexity it follows that if there exists a point, \( \tilde{z} \in N_r \cap L^n \), with \( f(\tilde{z}) < f(z^*) \) it implies that \( \nabla f(z^*_r)^T(\tilde{z} - z^*_r) < 0 \). But since it was assumed that the minimization resulted in \( \mu = f(z^*_r) \) we find that \( \nabla f(z^*_r)^T(z^* - z^*_r) = 0 \) for the active constraint. We may, therefore, conclude that there exists no point \( \tilde{z} \in N_r \cap L^n \) with \( f(\tilde{z}) < f(z^*_r) = f(z^*) \). Thus \( z^* \) is optimal also in \((P)\). □

A non-optimal point will, due to the pseudo-convexity, thus, always correspond to a solution with a \( \mu \)-value strictly lower than the value of the contour \( f(z^*_r) \). Thus, the only additional termination criteria required, besides the feasibility and under-estimation criteria (6) and (7), when incorporating a pseudo-convex objective function is the simple check \( \mu = f(z^*_r) \) or

\[
|f(z^*_r) - \mu| \leq \epsilon_f
\]

in practice. A situation with \( \mu = f(z^*_r) \) will always occur. This is due to the fact that the method produces a sequence of sub-problems with feasible regions, \( N_r \cap L^n \), of decreasing size (due to the strengthening of the reduction constraint) so in the limit we will clearly have \( \mu = f(z^*_r) \).

**COMPUTATIONAL IMPROVEMENTS**

The following improvements have been implemented to increase the computational efficiency of the method.

**Remove or replace linearizations of the objective function constraint.** When a new improved upper bound, \( f(z^t_{r+1}) \), of the objective function is found, a new problem \((P^1_{r+1})\) is defined and solved. The new problem \((P^1_{r+1})\) includes, in addition to the original linear constraint in \( L \), one linearization of the objective function constraint (1) in \( L^1_{r+1} \) as well as a new reduction constraint \( f(z) \leq f(z^t_{r+1}) \) defined in the set \( N_{r+1} \). Linearizations of the objective function constraint on previous level curves and linearizations of the reduction constraint at previous level curves have, in this case, been removed. This is an alternative solution strategy in the Alpha-ECP solver and can be defined by the PCO strategy in the ECP parameters window. However, old linearizations of the reduction constraint can be utilized in the subsequent iterations, since the reduction constraint is a pseudo-convex inequality constraint with any constant value of the RHS. Thus, when a new improved upper bound on the objective function is obtained, old linearizations of the reduction constraints can be modified to correspond to the new improved value of the RHS. This will improve the performance of the algorithm in many cases. This solution strategy can also be defined by the PCO strategy in the ECP parameters window. Linearizations of the objective function constraint (1) must, however, be removed, or may be modified to be cutting planes of the reduction constraint. When the replace strategy is used in the algorithm, old linearizations of the objective function constraint are, in fact, modified and replaced to be cutting planes of the reduction constraint.

**Line-search for the contour.** Assume that we have obtained points on the same contour \( r \), \( z^1_r, \ldots, z^n_r \), and that the next MILP solution, \( z_{\text{MILP}} \), does not satisfy the reduction constraint, i.e. \( f(z_{\text{MILP}}) > f(z^*_r) \). It is now possible to compute a point on this contour without solving the entire MINLP problem \((P^n)\) to optimality or feasibility. First compute the central point
\[
\bar{z}_{mid} = \frac{1}{n} \sum_{i=1}^{n} z_i^l \text{ of the points on the contour. Since } f(\bar{z}_{mid}) \leq f(z_i^l) \text{ (due to convexity of the levels-sets of } f) \text{ and } f(z_{MILP}) > f(z_i^l) \text{ (assumption)} \text{ a point on the line segment } [z_{MILP}, \bar{z}_{mid}] \text{ located exactly on the contour } f(z) = f(z_i^l) \text{ can be found using a line-search strategy. The objective can now be linearized at the relaxed point } \tilde{z}_{i+1} = \lambda \bar{z}_{mid} + (1-\lambda)z_{MILP} \text{ where } \lambda \text{ is obtained with the line-search strategy. Instead of solving } (P^n) \text{ as a MINLP problem we have “solved” it as a single MILP combined with a simple line-search. Observe, however, that due to the continuous nature of the line-search, the integer restrictions may be violated at } \tilde{z}_{i+1} \text{ and termination can not, in general, occur on a point obtained from the line search. }
\]

**Line-search for the minimum.** Suppose that a MILP solution \( z_{MILP} \) that violates the reduction constraint has identical integer part to one of the points on the contour, say \( z_i^l \) (\( y_i^l = y_{MILP} \)). Since the integer parts match, we can in this case find the minimum of \( f(z) \) on the line segment \( [z_{MILP}, z_i^l] = [x_{MILP}, x_i^l] \) instead of locating the contour. In this case the obtained point will be feasible, i.e. it satisfies all integer restrictions, since the minimization was performed in the continuous space only. This might improve the convergence on the continuous variables for certain problems. Also observe that the line-search for the minimum can always be done for NLP problems and this step is in fact similar to a step of the well-known NLP method “the method of feasible directions”. The line-search strategies can be combined or not, with the remove or replace strategies mentioned above. The strategy can be selected with the PCO Strategy in the ECP parameters window.

**THE ECP ALGORITHM**

**Step 0.** The procedure can be started in two different ways. In both alternatives a MILP problem is first created from the original MINLP problem by only considering the linear constraints. This problem is called \((P^0)\) (\( k=0 \) and \( J_0 = 0 \)). The upper bound of the objective function is defined as a large positive value \((10^{30})\). Then go to step 1 (this is the default strategy). Alternatively, an initial starting point, infeasible or feasible in \((P)\), can be given. If the starting point is infeasible in \((P)\) then an initial cutting plane or several initial cutting planes can be generated. On the other hand if the starting point is feasible in \((P)\) then a valid upper bound of the objective function is obtained and an initial linearization of the objective function can be used in the problem \((P^0_0)\). This latter alternative requires, however, that the objective function is defined as a pseudo-convex objective function with a pco-type constraint in the problem formulation and the Extended Q-C Strategy should be used in this case. The initial MILP strategy must in addition be defined. Then go to step 1.

**Step 1.** Solve the MILP problem \((P^k)\). If this is the first sub problem, \((P^0)\), there is a possibility of defining that this problem should be solved as a relaxed problem instead of a MILP problem. If the original problem \((P)\) is feasible then \((P^0)\) will also be feasible. If this is not the first problem, then \((P^k)\) can either be infeasible or feasible.

**Step 2.** If the MILP problem, \((P^k)\), to be solved in step 1 was infeasible, then go to step 3. If \((P^k)\) is feasible, then go to step 4.
Step 3. When the MILP problem, \((P^k)\), to be solved is infeasible, then modify all the \(\alpha_j^k\) parameters by using equation (4). Then let \(k = k + 1\), create a new MILP problem and go to step 1. This procedure continues until a feasible solution of \((P^k)\) is found in step 1. If the problem \((P)\) is feasible, then a feasible solution will eventually be obtained for sufficiently large \(\alpha_j^k\) values.

Step 4. Here we test if the value of the objective function is less than or equal to the best upper bound obtained so far. If the value is less than or equal to the upper bound, then continue at step 8.

Step 5. If the value of the objective function was greater than the value of the best upper bound, then it is possible to use some of the line-search strategies to improve the solution point. Depending on the line search strategy, line search may not performed or performed to the minimum value of the objective function or to the value of the actual contour on the line between the actual point and the central point obtained from previous points on the contour. The strategy used can be defined in the \textit{ECP parameters} window. Continue at step 6.

Step 6. In this step we repeat the test made in step 4, for the point obtained at step 5. If the value of the objective function is greater than the value of the best upper bound obtained so far, then the reduction constraint is not satisfied. Then go to Step 7. If the value of the objective function is less than or equal to the value of the previously used upper bound on the objective function, then continue at step 8.

Step 7. Here the value of the objective function is greater than the value of the best upper bound obtained so far and the reduction constraint is not satisfied. In this case we will make a linearization of the reduction constraint and add it to the previous MILP problem. Let \(k = k + 1\) and and go to step 1.

Step 8. In this step the values of all non-linear constraints are calculated. The values of the non-linear constraints at this point are given by, \(g_i(z_k), i = 1, \ldots, m\). The largest value of any of the non-linear functions will be defined as \(g^{k}_{\text{max}}\). Now continue at step 9 or step 10. These alternative strategies can be defined in the \textit{ECP parameters} window. Step 9 is not required to ensure global convergence for pseudo-convex problems but this step may help to improve the convergence properties of the algorithm, especially for non-pseudo-convex problems. Step 9 is always satisfied for convex problems.

Step 9. If we require the linear functions to be so-called \textit{local under-estimators}, then at each solution point the following inequalities,

\[
l_j(z_k) \leq \tilde{g}_j(z_k)
\]

need be satisfied for all linear constraints and corresponding non-linear constraints. \(i\) is the index of the non-linear constraint corresponding to the linear constraint \(j\). \(l_j(z_k)\) is the value of the linear constraint and \(\tilde{g}_j(z_k)\) is the value of the corresponding non-linear constraint at this solution point. When (9) is to be evaluated, there are some alternative strategies to be used. We can either evaluate (9) only at MILP solutions which are epsilon-infeasible solutions of the corresponding MINLP problem (the \(g_{\text{max}}^{k}\) value is greater than \(\epsilon_g\), where \(\epsilon_g\) is a small tolerance) or always evaluate (9) (which means an evaluation even at epsilon-feasible MINLP solutions). These two alternative strategies to ensure the linear functions to be \textit{local under-estimators} can be defined in the \textit{ECP parameters} window. If (9) is not satisfied for all linear constraints then
the $\alpha^k_j$ parameters are modified for those linear constraints not satisfying (9) by using (4 or 5). If the solution from the MILP problem is epsilon-feasible for the MINLP problem then the $\alpha^k_j$ values are updated by (5) and if the point is an epsilon-infeasible solution to the MINLP problem, then the $\alpha^k_j$ values are updated by (4). If any of the $\alpha^k_j$ values are updated, let $k=k+1$. Then create a new MILP problem and continue with step 1. If (9) is satisfied for all linear constraints continue with step 10. Observe that inequality (9) is automatically satisfied after problem $(P^0)$, since $J_\infty=0$. In addition, inequality (9) is always satisfied for convex constraints. Also observe that inequality (9) means that the linear function should underestimate the corresponding non-linear constraint in all solution points (the definition for the linear functions to be so-called local under-estimators). The use of local under-estimators is, however, not necessary in order to ensure global convergence for a MINLP problem $(P)$. However, the use of step 9 has been found to improve the performance of the algorithm on certain non-convex MINLP problems, therefore this step is included as an option in the algorithm. The alternative strategies to obtain local under-estimators can be defined in the ECP parameters window.

**Step 10.** Now start to check for termination. Step 10 is the first of the different termination criteria used. We will first test for feasibility. A point is considered $\epsilon$-feasible in $(P)$ if Eq. (6) is satisfied. If the objective function is linear and the inequality constraints are convex and (6) is satisfied then if in addition the MILP solution was an optimal MILP solution the procedure will terminate (after testing the additional criterions) and the solution point is the global optimal solution to the corresponding convex MINLP problem. If the problem is non-convex and (6) is satisfied then the procedure continues at step 12. If (6) is not satisfied, then continue at step 11.

**Step 11.** If any $g_i(z_k) \geq 0$, then linearize one or several non-linear constraints and add the linearized constraints to the previous MILP problem. At this point some alternative strategies can be used. The default strategy is that only one linearization is added to the MILP problem. The linearization is made on the non-linear constraint which violates the most. As an alternative, linearizations for all non-linear constraints satisfying $g_i(z_k) \geq 0$, can be added to the following MILP sub problem. If the non-linear constraint $(g_i(z_k) \geq 0)$ is a general non-linear constraint then the criterion $g_i(z_k) \geq 0$ above is replaced by $g_i(z_k) \geq \epsilon g_i$. One can also add linearizations for a non-linear constraint in each iteration if the non-linear constraint is convex. If linearizations are added, then update the number $J_k$ with the number of new linearizations. Observe that only the $\tilde{g}_j = g_i(z_k)$ value and the derivatives $\nabla \tilde{g}_j(z_k)$ on each linearized constraint need be stored to the next iterations, not $z_k$. $\alpha^k_j=1$ on new linearizations. Then let $k=k+1$ create a new MILP problem and go to step 1. The different solution strategies in step 11 can be defined in the ECP parameters window.

**Step 12.** Since the MILP solution point, $z_k$, has passed step 4 or step 6 the value of the objective function at step 12 is either less or equal to the best valid upper bound on the objective function obtained so far. If the value of the objective function at the MILP solution point, $z_k$, is strictly less than the best valid upper bound then go to step 13 otherwise continue at step 14.

**Step 13.** Here the value of the objective function is strictly less than the best valid upper bound obtained so far. In this case we will update the new improved upper bound as well as the reduction constraint in $N_{r+1}$. In addition we will add a linearization of the objective function
constraint to the set $L^1_{r+1}$ and remove the old linearizations of the objective function constraint from the set $L^1_{r+1}$. Depending on the strategy defined in the ECP parameters window we can completely remove old cutting planes of the reduction constraint, in the $\Omega_{k+1}$ set, or utilize the previous information and replace the old cutting planes with cutting planes corresponding to the new reduction constraint (with the new improved value of the objective function). Also the old linearizations of the objective function constraint can be utilized by replacing them to be cutting planes of the new reduction constraint. Let $k=k+1$ and $r=r+1$ and go to step 1.

**Step 14.** Here we will test for the second termination criteria. If any of the non-linear inequality constraints are pseudo-convex then in addition to criterion (6) also criterion (7) must be satisfied for all corresponding linear constraints at termination. In addition the MILP problem solution should, of course, be optimal at final termination. This is tested later on at step 16 for all strategies. If any of the $\alpha^k_j$ values are not big enough, the corresponding cutting plane is not a valid cutting plane and there is a risk that a part of the feasible region of the problem $(P)$ is cut off. If this is the case then continue at step 15.

If each $\alpha^k_j$ value is sufficiently large, i.e. satisfies Eq. (7) and the objective function is linear then this will be the global optimal solution of the problem $(P)$ if the MILP solution was obtained as an optimal MILP solution. Continue to step 16 and test the MILP solution. Step 14 is always satisfied for convex problems.

An alternative termination criterion for quasi-convex NLP problems
As an alternative to Eq. (7) a weaker termination criterion can be applied to NLP problems with quasi-convex inequality constraints. In Still and Westerlund (1997) it is shown that only the active constraints (constraints equal to zero at termination) need to be outer approximations of the feasible region of $(P)$ for NLP problems. In Still and Westerlund (1997) it was shown that convergence for NLP problems will be faster using the alternative criterion instead of criterion (7). In this case only the active cutting planes at the current iteration point need be outer approximations of the feasible region of the problem. However, it should be pointed out that termination to the global optimal solution is, in general, only ensured for NLP and not for MINLP problems with this alternative criterion. Thus, for MINLP problems, criterion (7) must still be used.

**Step 15.** In this step we ensure the modified linear constraints to be feasible under-estimators, i.e. valid cutting planes. For the cutting planes to be feasible under-estimators the inequalities given by Eq. (7) should be satisfied for all $j$. The cutting planes are considered feasible under-estimators or valid cutting planes, underestimating the entire boundary, or more precisely, the convex hull of the feasible region of $(P)$ if the $\alpha^k_j$ values in each function, $l_j(z)$, satisfies criterion (7). For general non-linear constraints $h = \varepsilon_h$ is also used in the algorithm. In this case we cannot, however, generally ensure that the corresponding cutting planes will be feasible under-estimators. If criterion (7) is not satisfied, for some cutting planes, then modify the $\alpha^k_j$ parameters in the corresponding function, $l_j(z)$ . Here some alternative strategies can be used.

The $\alpha^k_j$ values may be updated every time when criterion (7) is not satisfied or the $\alpha^k_j$ values updated only at epsilon-feasible solutions of the corresponding MINLP problem. As was mentioned previously, a solution is considered epsilon feasible in $(P)$ if the maximal function value, $g^\varepsilon_{max}$, of any of the non-linear constraints is less than or equal to $\varepsilon_g$, where $\varepsilon_g$ is a given tolerance. If the solution of the MILP problem is an epsilon-feasible solution to the
corresponding MINLP problem, then the \( \alpha_j^k \) values are updated by (5) and if the point is an epsilon-infeasible solution to the MINLP problem, then the \( \alpha_j^k \) values are updated by (4). These alternatives strategies ensure the linear functions, \( l_j(z) \), to be feasible under-estimators can be defined in the parameters menu. Then go to step 1.

**Step 16.** Here we make a test if the MILP solution obtained was an optimal MILP solution. If the criterion is satisfied then continue at step 18.

**Step 17.** If the criterion at step 16 was not satisfied then change the MILP solutions strategy. If the MILP solution was stopped at an integer feasible (but not optimal) solution then increase the MIP Solutions Limit parameter by one and continue at step 1.

**Step 18.** Here we make the final test for termination. This test is made only in case we use the Extended Q-C strategy, (and we have defined a pseudo-convex objective function constraint with the pco-type definition). If criterion (7) is satisfied, then this is the global optimal solution of the MINLP problem \((P)\). On the other hand if the \( \mu \) value is strictly less than the best valid upper bound of the objective function obtained so far, then continue at step 19.

**Step 19.** Here let \( n = n + 1 \) and add a new linearization of the objective function constraint to the set \( L_n^\alpha \). Then continue at step 1.

**Step 20.** Here all termination criteria have been passed and the solution is the global optimal solution to the MINLP problem \((P)\).

Figure 1. Flowchart of the Alpha-ECP Algorithm
GETTING STARTED
The necessary files and the setup procedure of the $\alpha$-ECP program are all stored in the PackageXP.zip or Package2000.zip file. By unzipping the files and, thereafter, by clicking on the setup file the files will be stored (as default) in the directory: C:\ProgramFiles\AlphaECP. Some sample problem files with corresponding parameter files will be stored in the sub-directory: C:\ProgramFiles\AlphaECP\Problems. Different documentations will be stored in C:\ProgramFiles\AlphaECP\Documents.

HARDWARE AND SOFTWARE REQUIREMENTS
The present version 5.101 of the $\alpha$-ECP program is written for Windows NT, 2000 and XP. It is a prerequisite that the PC has at least 16Mbyte memory and 20 Mbyte free space on the hard disk. The $\alpha$-ECP program uses a commercial MILP solver, CPLEX from CPLEX Inc., solving the MILP sub problems. The version of the CPLEX callable library should be 7.0 or newer. In addition, the program needs a 32-bit FORTRAN-90 compiler producing DLL files from the generated program files. Compaq Visual FORTRAN (http://www.polyhedron.com/vf/vfortran.html) may for example be used. Both CPLEX and FORTRAN-90 licenses are therefore needed to run the $\alpha$-ECP program.

The Problem Code Generator and the $\alpha$-ECP-algorithm are written in FORTRAN-90 © Tapio Westerlund. The interface is written in Visual Basic 6.0 © Kurt Lundqvist. The actual version of the program can be found by clicking on the About AlphaECP… sub menu in the Help menu in the main ECP window. The following window will then be displayed:

![About Window](image)

THE $\alpha$-ECP PROGRAM
Start by unzipping the installation files into a temporary directory. Then doubleclick on the setup icon and follow the instructions. The AlphaECP program will then, by default, be installed in the C:\ProgramFiles\AlphaECP directory. In addition the dynamic link library file containing the CPLEX
Callable library should be copied from the CPLEX directory to the `C:/ProgramFiles/AlphaECP` directory under the file name CPLEX.DLL.

Once the program is installed, the program is started by double-clicking on the `Alpha-ECP.exe` icon. The Alpha-ECP window will then be displayed. In the main ECP window several menus can be found. The different functions of the program are changed and the program operated from the main window. The three first menus `File`, `Edit` and `Search` have instructions for opening, editing and printing problem files. From the last `Help` menu this manual can be found by using the `Documentation` sub menu. The actual version of the solver can be found, as was shown above, from the sub menu `About AlphaECP…` in the `Help` menu. Other menus are: `View`, `Settings` and `Parameters`. Later on in this user’s guide the function of these menus are described.

Once the main window is opened, the `File` menu is usually the first selected menu. From this menu one can operate the problem files. The main ECP window and the `File` sub menu is displayed below.

If the problem is already written as a problem file, this file can be opened and imported to the ECP program by selecting the sub instruction `Open…` as illustrated above. The `Open…` instruction will then open a new window showing all the problem files. Problem files are written with a default extension `.ecp`. The actual problem file can then be selected from the `Open` window.

The `Open` window is shown below.
Once a problem file has been selected, the file is imported to the main ECP window. The content of the file can thereafter directly be edited in the main window. Different file manipulation instructions are also found in the Edit and Search menus. The Edit menu has the following sub instructions: Cut, Copy, and Paste. The Search menu has the following sub instructions: Find and Find next.

**The problem file**

The whole MINLP problem is written in a problem file. The syntax of the linear part of the problem follows the syntax used in standard LP files. LP-files can be used, for example, in the interactive CPLEX MILP solver. Non-linear constraints cannot, however, be written in standard LP-format. We have, therefore, introduced an extended LP format where non-linear constraints can be written using FORTRAN-90 syntax. This means, for example, that a multiplication is defined by the character ‘*’ in the non-linear constraint expressions. Since this may result in an inconsistency between the syntax used in the linear and non-linear parts of the problem, it is also allowed to use ‘*’ in the .ecp problem files to define multiplication between a constant and a variable in the linear part of the problem. The non-linear constraints can be defined in a section started by the section statement Non-linear constraints. The non-linear constraints section of the problem must be written after the linear constraints have been defined.

**NB:** All variables involved in the MINLP problem must be defined in the linear part of the problem. Thus for example, a constraint c: can be included instead of defining a bound on the variable in the Bounds section if the variable is not defined in the linear part of the problem.
In the following, the MINLP problem found in the problem file `prob01.ecp` is displayed.

**The non-linear constraints**

After the section statement *Non-linear constraints* has been written, the non-linear inequality constraints are defined. Each non-linear inequality constraint is defined by the following syntax:

```
gtype string : expression operator rhs
```

*type* is used to define the type of non-linear constraint. Four types are available. If *type=*`cvx` then the constraint is considered as a convex constraint and if *type=*`qcv` then the constraint is considered as a quasi-convex constraint. If *type=*`pco` then the constraint is considered as an objective function constraint of the type in Eq. (1) and the objective function is considered to be pseudo-convex. If *type=*`coc` then the constraint is considered as a convex objective function constraint, and the strategy given for pseudo-convex objective functions can be used. Observe also that the objective function expression must be followed by the objective function variable. Only one `pco` or `coc` type constraint can, of course, be defined. If no *type* is defined then the constraint is considered as a general non-linear constraint in the algorithm. Some of the examples given later on, illustrate the use of the different constraint types.

The *string* is of suitable form and is used only to identify the constraint. A non-linear inequality constraint is defined as an *expression* written by using FORTRAN-90 syntax. By default the computations are made in double precision, thus *double precision standard functions* must be used. A line must be shorter than or equal to 256 characters, but the expression can continue on following lines. The end of the expression is defined by an *operator*.

The *operator* is either “<=” or “>=”
Finally the right hand side $rhs$ is defined ($rhs$ is typically 0). Observe that the RHS is moved to the LHS of the expression in the calculations. Thus it is recommended that the RHS is equal to zero already when defining the expression. $Rhs$ must be shorter than or equal to 57 characters. The number of non-linear constraints and the number of variables are only restricted by the available memory size. The variable names are, however, restricted to be shorter or equal to 16 characters.

Once a non-linear constraint has been defined it can be followed by either:

1) a new non-linear constraint
2) a default value statement for the derivatives
3) analytical expressions for the derivatives
4) a definition that a derivative should be calculated numerically
5) or followed by some of the section statements:
   \textbf{Bounds}, \textbf{Integers}, \textbf{Binaries} or \textbf{End}.

Each non-linear constraint can be followed by expressions defining how the derivatives should be calculated. If analytical expressions are available they can be defined. If no other expression than the non-linear constraint itself has been defined, then the derivatives are calculated numerically by a forward difference approximation. The step size, $\Delta$, in the forward difference approximation can be modified using the sub instruction \textit{Other} in the \textbf{Parameters} menu.

\textbf{Default values for the derivatives}
In many problems containing a large number of variables it is useful to define the default values of the derivatives in such a way that undefined derivatives need not be calculated numerically. In the problem file \textit{default} values of the derivatives can be defined as follows:

\begin{verbatim}
Default : expression
\end{verbatim}

\textit{expression} is an analytical expression written in FORTRAN-90 syntax (usually the expression is a constant equal to 0). The default expression must be shorter than or equal to 60 characters. The default statement can be written before or after the other specified expressions of the derivatives. \textit{Observe that derivatives of a function must be calculated with respect to all variables, therefore the default definition may speed up the calculations since unnecessary numerical derivative calculations can, thus, be avoided.}

\textbf{Analytical expressions for the derivatives}
If analytical expressions of the derivatives are known they can be defined by the following syntax:

\begin{verbatim}
dg string /d variable : expression
\end{verbatim}

The \textit{string} can be of suitable form, but it refers to the previous non-linear constraint. \textit{variable} is the variable name and \textit{expression} is the analytical expression of the derivative. The expression must be written in FORTRAN-90 syntax and a line must be shorter than or equal to 256 characters. The expression can, however, be continued on the following lines. \textit{Observe that the derivatives are always defined for a function $g(z) \leq 0$ even though the non-linear constraint itself has been defined using the operator “$\geq$”}.

\textbf{A definition that the derivative should be calculated numerically}
If it is difficult to obtain analytical expressions of all derivatives, then it is possible to specify that some derivatives should be calculated numerically. The syntax for giving this information is as follows:
SOLVING THE MINLP PROBLEM

In order to solve a MINLP problem the following buttons in the main ECP window are used:

- **Optimize**
- **Continue/Step**

When the **Optimize** button is pressed the problem in the main window will be optimized. Intermediate results are shown in the lower right corner of the window and solution results can be displayed or stored in files after the termination or at intermediate iterations. When the Optimize button is pressed the AlphaECP program first calls the problem code generator *alpha0* and thereafter the produced complete ECP solution procedure *alpha* as well as the MILP solver CPLEX repeatedly during the solution phase. The *alpha0* code generator produces necessary solution code to produce the code called *alpha* by which cutting planes are generated during the solution phase while the MILP solver solves the intermediate MILP problems.

Once the **Optimize** button has been pressed, the ECP program code generator (*alpha0.dll*) automatically generates three FORTRAN-90 sub program files: *alpha.for*, *alpha2.for* and *alpha3.for*. These programs are then automatically compiled and linked together with the ECP solver (the ECP solver is in the *alpha1.obj* file) producing a dynamic link library. When the program code generator is running and the sub programs are produced, **Running alpha0**, is displayed in the lower right corner of the main ECP window.

The *alpha.for* subroutine which is produced by the alpha0 program will call the main callable functions in the ECP solver. The file *alpha2.for* contains a FORTRAN-90 sub program able to compute the non-linear inequality constraints and their corresponding analytical derivatives. (As an example the *alpha2.for* file for the first considered example problem is given in appendix B). The *alpha3.for* file contains the sub program to compute numerical derivatives. The compiling and linking is done automatically and a **LINK** window is displayed when this is done. Only the source code of *alpha2.for* remains on the directory \*C:\Program Files\AlphaEcp* after the *alpha.dll* dynamic link library file has been produced.

In the **LINK** window error messages can be found if the expressions used when defining the non-linear constraints have not followed the syntax. If problems occur, one can also open the *alpha2.for* file or try to compile it separately in order to get more information about what is probably wrong. Error messages (***) may also be found in the *alpha.nlnc* file.

If the **Optimize** button is pressed another time the optimization can start directly, no new programs are generated and no new DLL files are produced, unless the problem file in the main ECP window has been changed. If the problem in the main ECP window has been changed, then the “Optimize” button acts in the same way it does the first time it is pressed. The problem in the main ECP window is stored (as default) in the *alpha.ecp* file. The name of this file can be changed from the main ECP window.

The **Stop** button can be used to stop the program, for example, to change some parameters in the ECP solver or in the CPLEX MILP sub problem solver. This can also be done in order to restart solving a problem.
When solving a problem, the number of master iterations (when cutting planes are generated) as well as the total number of iterations is displayed in the main ECP window. In addition the cumulative number of optimal MILP solutions is displayed. If all intermediate MILP problems are solved to optimality, this latter number is equal to the total number of iterations.

**NB:** If the non-linear constraints cannot be written in explicit form, in the problem file, it is possible to define **Allow modified FORTRAN files** in the **Settings** menu. The program will then stop (after the 'start new problem' button has been pressed) before linking the programs together allowing the user to edit the generated alpha2.for file. See for example, example problem prob08.ecp in the appendix.

When the **Optimize** button is pressed the first time and the DLL files are produced the following window will be displayed:

The command line to compile and link the DLL is, by default, generated into a command file **alpha.cmd**. The content of this file, alpha.cmd, should be changed if another than the FORTRAN-90 compiler and linker shown in the window above is used. The command line can be changed by the **Fortran Compiler Syntax** instructions in the **Settings** menu.

Both the ECP algorithm and the MILP solver use a set of different parameters when solving the MINLP problem. These parameters can be changed from the main ECP window by using the **Parameters** instruction. In addition to the parameters in the ECP and MILP solver, some other parameters can be manipulated by the **Parameters** instruction. These parameter settings include the step size used in forward difference approximation when calculating numerical derivatives. (The step size can be changed in the **ECP Parameters** window by first using the **Parameters** instruction). Three default parameter settings instructions are available: **Convex, Quasi-Convex** and **Extended Quasi-Convex**.

The **Parameters** window is shown in the following:
From the sub menu **ECP Parameters** different parameters of the ECP method can be changed as can be seen from the window above.

By using the **Ecp Linearization strategy** instruction two different solution strategies for adding linearizations can be selected. The first one is the default strategy. In this strategy only one linearization is added (linearization of the non-linear constraint with the largest (and non-negative) function value). With the second strategy, linearizations of all non-linear constraints being greater than zero are added at the master iterations (for general non-linear constraints, linearizations are added only if the value of the non-linear constraint is greater than the given tolerance, $\varepsilon_g$).

Using the **Starting with** instruction, the procedure can be started either by solving a MILP problem or a relaxed problem.

Using the **Starting Point** instruction, one can start the procedure from a user supplied starting point. In this case the program asks for a starting point to be given by the user. The starting point can be given as a file.

By using the **Alpha strategy** instruction the alpha-updating strategy is selected. In previous versions of the solver different strategies could be used. In the actual version the strategies are defined by the overall default strategy. If convex defaults are used, then constant alpha values equal to 1 are used. For the other overall default strategies the alpha values are updated by equations (4-5) using given $\beta$ and $\gamma$-values. When the alpha-values are updated and the linear functions (3) are added as inequalities to the problem ($P^k$), the functions can be written such that only the RHS of the inequality is changed by the alpha-value or such that both LHS and RHS are changed by the alpha-value. When using the alpha-updating procedures the linear inequality constraints are written such that only RHS is changed by the alpha value in version 5.01 of the solver.
Each linear inequality constraint, \( l_j(z) \leq 0 \), is produced from a given non-linear constraint and the linear function, \( l_j(z) \), in principle, defined by Eq. (3). However, since the index, \( j \), only counts for the number of cutting planes, one cannot afterwards identify the corresponding non-linear constraint and the number of times the \( \alpha_j^k \)-value has been updated. Therefore we have used a different syntax in the algorithm to define the linear inequality constraints, \( l_j(z) \leq 0 \). A linear constraint where only the RHS is changed when updating the alpha-value is written as,

\[
\text{cg } i\_u\_k: \ \nabla \tilde{g}_j^T \cdot z \leq \nabla \tilde{g}_j^T \cdot z_k \frac{-\tilde{g}_j}{\alpha_j^k} \tag{10}
\]

In prior versions of the solver a linear inequality constraint where both the RHS and the LHS could be changed when updating the alpha-value was written as,

\[
\text{cg } i\_u\_k: \ \alpha_j^k \cdot \nabla \tilde{g}_j^T \cdot z \leq \alpha_j^k \cdot \nabla \tilde{g}_j^T \cdot z_k - \tilde{g}_j \tag{11}
\]

By the index \( i \) the corresponding non-linear constraint is defined. The index \( u \) counts the number of times the \( \alpha_j^k \)-value has been updated and \( k \) counts the total number of iterations.

Using the \textit{Beta} instruction the \( \beta \) value in equation (4) can be changed.

Using the \textit{Gamma} instruction the \( \gamma \) value in equation (5) can be changed.

The \textit{Beta} and \textit{Gamma} values can only be changed when using the \textit{Quasi-Convex} or \textit{Extended Quasi-Convex} strategy.

Using the \textit{Local Underestimator} instruction, one can select whether step 9 in the algorithm is then used or not. There are two alternatives if any local under-estimators are used. The local under-estimator check (inequality (6)) can be used to update the alpha-values, either only at MILP solutions which are epsilon-infeasible for the corresponding MINLP problem (\( g_{ \text{max}}^k \geq \varepsilon_{\text{e}} \)) or both at solution points which are epsilon-feasible and epsilon-infeasible for the corresponding MINLP problem. The first updating strategy is used as default when defining extended quasi-convex defaults. The local under-estimator step is neglected when defining quasi-convex defaults. Convex functions always satisfy criterion (6).

Using the \textit{Feasible Underestimator} instruction, one can select how step 14 in the algorithm is performed. There are two possibilities if any alpha-updating procedure is used. This step is neglected when using convex defaults, since convex functions always satisfy the inequality (7). The feasible under-estimator check, inequality (7), for quasi-convex is used to update the alpha-values, either only at MILP solutions which are epsilon-infeasible for the corresponding MINLP problem (\( g_{ \text{max}}^k \geq \varepsilon_{\text{e}} \)) or both at solution points which are epsilon-feasible and epsilon-infeasible for the corresponding MINLP problem. The first updating strategy is used as default when defining quasi-convex defaults. For general non-linear constraints the same procedure as for quasi-convex constraints is used.

Utilizing the \textit{PCO Strategy} instruction different strategies when using a pseudo-convex objective function can be defined. Different strategies can only be selected in \textit{Extended Quasi-Convex} mode after the \textit{Extended Quasi-Convex strategy} has first been defined. By the \textit{PCO Strategy} old objective function constraints can be removed or replaced and line-search strategy can be combined with the mentioned remove or replace strategy. The \textit{line-search} can either be performed to the \textit{contour} or to the \textit{minimum} objective function value on the line between the actual point and the so-called central point. The central point is calculated from the old solution points on the same contour.
Using the *ECP Iteration Limit* instruction the maximum number of MILP iterations before termination can be specified.

Using the *ECP Termination Criteria* instruction, the termination criteria (6), (7) or (8) can be selected. Observe that criterion (6) is already enough for convex MINLP problems and in addition with (7) for MINLP problems with a linear objective function and quasi-convex constraints. Eq. (8) is needed if in addition, the objective function is pseudo-convex. In all cases the final MILP solution should, of course, also be optimal.

With the *Epsilon_g* instruction the value of the $\epsilon_g$ tolerance in Eq. (6) can be changed.

With the *Epsilon_h strategy* instruction the value $h_j$ in the denominator of Eq. (7.a) can be changed either to be given as $h_j = \epsilon_h = \epsilon_z$ or as $h_j = \epsilon_z \cdot \sqrt{V g_j^T V g_j}$ according to Eq. (7.b).

With the *Epsilon_f* instruction the value of the $\epsilon_f$ tolerance in Eq. (8) can be changed.

In addition to the main *ECP Parameters* it is possible to specify the step size used in the forward difference approximation when calculating numerical derivatives, by using the *Delta* sub instruction in the *ECP Parameters* menu.

From the *Parameters* menu different parameters of the MILP solver in the CPLEX callable library (the Branch and Bound method) can be changed by using *Cplex Parameters* as shown below.

All MILP solver parameters act independently, except for one parameter: *MIP Limits Solutions*.

As previously mentioned, intermediate MILP problems need not be solved to optimality in the ECP algorithm. Intermediate MILP problems may be solved to an integer feasible solution or only to an integer relaxed solution. Only to verify an optimal solution the final MILP problem must be solved to optimality. By using the CPLEX parameter *MIP Limits Solutions* it is possible to solve intermediate MILP sub problems only to an integer feasible solution. If the *MIP Limits Solutions* parameter is initially put, for example, to one then the first MILP solution is terminated after the MILP solver has found the first integer feasible solution. The *MIP Limits Solutions* parameter is thereafter increased by one, according to the flow sheet in figure 1. The following MILP sub problem(s) is (are) then solved to the second integer feasible MILP solution and so on. Termination takes place when the MILP solution is verified to be optimal and not interrupted by the value of the *MIP Limits Solutions* parameter.

The *MIP Limits Solutions (initial)* parameter can be used to solve intermediate MILP problems only to integer feasible solutions. It should be observed that a ECP iteration may in this case only be to solve the same MILP problem from one integer feasible solution to a following (better) integer feasible solution. In some examples the solution of intermediate MILP problems to integer feasible solutions makes the algorithm considerable faster. An example problem is given in appendix E. It is found that only one MILP problem was solved to optimality in order to solve the convex MINLP problem in prob12.ecp to optimality, by using this strategy. In the example the *MIP Limits Solutions* parameter was given an initial value equal to 1. From the active parameters window, for this example, shown in appendix E it can be found that the *MIP Limits Solutions* parameter obtained finally a value equal to 7 at the optimal solution.
The parameters which are defined in the Parameters window can be loaded from or saved in a parameter file by using the Load Parameters from file and Save As ... buttons. The parameters are, as default, saved in a parameter file with the same name as the problem file, but with the extension .par and the files are stored in the same directory as the problem files. The default directory is:

`C:\Program Files\AlphaEcp\Problems`

Different user selected settings can also be defined. These settings can be changed by using the Settings menu. The sub instructions when using the Settings menu are illustrated below.
The instruction **Font Size** is used to change the font size used to display the problem file in the main ECP window.

The instruction **Allow Modified Fortran Files** can be used if the non-linear constraints cannot be written, in explicit form. If this instruction is used, the program stops after the ‘Start new problem’ button has been pressed and allows the user to modify the FORTRAN-90 sub program alpha2.for, by which the non-linear constraints are to be calculated, before the calculations continue.

The **Fortran Compiler Syntax** instruction can be used to change the command line for compiling and linking if another than the default Fortran compiler is used.

By the **Reserve Memory Size** instruction the available memory size, for intermediate storage of the data for the linearized constraints can be changed. The default memory allocations value is 50000. If the problem needs more memory than reserved by this instruction, then additional data is stored in the files alpha2.dat and alpha3.dat. Allocating memory speeds up the calculations. Note, however, that if a solution is to be continued, for example, after the power of the computer is turned off, then all the data for the linearized constraints should be saved in the mentioned files.

The **Memory Update Interval** instruction is active only if the reserved memory size is put to zero and the above mentioned intermediate data is stored in the alpha2.dat and alpha3.dat files. The memory update interval reduces the memory requirement when it is activated. The default value is 100.

The **Save CPU-times** instruction can be used to save the CPU-times used when solving the problem. The CPU-times are stored in the file alpha.cpu and the total CPU-time is displayed in the Solution window.

The **Save Objective and Variable Values** instruction can be used to save the objective function value and the variable values at different iterations when solving the MINLP problem. These values are stored in the file alpha.val.
The **Save Intermediate Results** instruction can be used to save intermediate results when solving the problem. The value of the objective function of the MILP sub problem, max Alpha and max g() values as well as the number of non-linear constraint and derivative evaluations at each iteration, \( k \), are stored in the file `alpha.int` if this instruction is active.

The **Display Solution Status** instruction is used to display the solution status when solving the MILP sub problems. When the display solution status instruction is active, solution messages from the MILP sub solutions are reported. Solution messages are only reported if all termination criteria in the CPLEX MILP solver have not been satisfied when the actual MILP sub problem was solved. The **Display Solution Status** instruction is by default, inactive.

The main ECP window has a further **View** menu. This menu has two sub instructions, **Solution Values** and **Active Parameters**. By using these instructions, the obtained solution values and the used parameter settings can be displayed.

When the **Solution Values** instruction is used a **Solution values** window is displayed. As an example, the solution values to the first given example are displayed as follows:

The solution can also be printed by pressing the **Print...** button and saved in a file by pressing the **Save As...** button.

The active parameter values can be displayed by using the instruction **Active Parameters**. In the following **Active Parameter Values** window, the parameters used when solving the previously given example problem are displayed:
The parameters can also be printed by pressing the **Print...** button.

**OBTAINING THE SOLUTION**

Once the **Optimize** button has been pressed the program solves the MINLP problem. The program can be stopped by pressing the **Stop** button, for example, to change different parameters in the algorithm. The procedure can then be continued by pressing the **Continue/Step** button. If the **Continue** button is pressed the procedure continues until some of the termination criteria is activated. If the **Step** button is pressed the procedure continues one iteration step. By clicking on the smaller problem window the linear problem at hand ($P_k$) can be displayed. The linear problems ($P_k$) are (as default) stored in a file with the name *alpha.lp*. The name of this file can, however, be changed from the main ECP window. The content of the main ECP window is shown below, after the first example has been solved. During the calculations **Running Alpha** or **Running CPLEX** is displayed in the lower right corner of the main ECP window. When **Running CPLEX** is displayed then the corresponding MINLP problem is solved and when **Running Alpha** is displayed then the calculations in the ECP algorithm, including function evaluations and derivative calculations, are made. The window also includes iteration index counters. **Master iterations** correspond to the number of MILP iterations where new linearizations have been added. **Total** is the total number of MILP problems solved and **Optimal MILPs** is the number of MILP sub-problems that has been solved to optimality. $Z_{\text{min}}$ is the actual objective function value of the
solved MILP problem and max $g(i)$ corresponds to the function value of the non-linear constraint $i$ with the largest function value at the actual iteration. $maxAlpha$ is the maximal value of the $\alpha_i^j$ parameters at the actual iteration.

Once the MINLP problem has been solved a Solution window is displayed. If termination criterion (6) is used, convergence to the global optimal solution is ensured only for convex problems. If both the termination criteria (7) and (8) are satisfied, then convergence to the global optimal solution for a pseudo-convex MINLP problem (P) is ensured.

When the program has terminated, the Solution window indicates that the program has terminated. In the solution window, termination=1 is displayed if the convex termination criterion (6) is satisfied, termination=2 if the quasi-convex NLP termination criterion is also satisfied or termination=3 if the termination criterion (6) together with (7) is satisfied and termination=4 if all termination criteria, (6), (7) and (8) are satisfied.

The solution values can thereafter be displayed by using the Solution Values instruction in the View menu in the main ECP window. Information about the solution can also be found directly from the main ECP window. At this point the main ECP window, for the example under consideration, looks as follows:
The final MILP sub problem, $P^K$, can be found in the smaller problem window as illustrated above, as well as in the created alpha.lp file.

**ANOTHER EXAMPLE**

The example given previously was a convex MINLP problem. In order to illustrate the necessity of using quasi-convex parameters in the ECP algorithm on problems with quasi-convex constraints, there follows an INLP problem with one quasi-convex inequality constraint. In the main ECP window below, the problem has first been solved using convex parameters.

As can be seen from the main ECP window, the solver has terminated in two iterations. The value of the objective function being equal to 11 at this feasible (but non-optimal !) point, (the solution values can be found by using the Solution Values instruction in the View menu. At this point the solution values are: $y_1 = 1$ and $y_2 = 4$). One master iteration has been made and the cutting plane at the first iteration has, in fact, cut off the optimal solution ($Z_{min}=10$).

Since Convex Defaults parameters were used when solving the problem, the algorithm has not checked if the cutting planes obtained from the non-linear constraint are feasible under-estimators of the boundary (or convex hull) of the feasible region of the quasi-convex INLP problem. The algorithm can thus terminate at a feasible but non-optimal point, unless the quasi-convex termination criterion is satisfied.
However, in order to solve this problem properly, the Convex Defaults parameters were changed to Quasi-Convex Defaults from the Parameters menu in the main ECP window. When this was done, the problem was resolved. It is then found that the algorithm uses 3 masters, and in total, 10 iterations. The value of the objective function at the termination point becomes the global optimal solution $Z_{\text{min}}=10$. The value of the non-linear constraint at this feasible and optimal point is -0.5 (as it was before). However, in this case all cutting planes in $P^k$ were feasible under-estimators. From the main ECP window it was also found at termination, that the maximal $\alpha^j$-value was equal to 32.

The main ECP window at this point (including ECP parameters from the Parameters menu) is shown below:

![ECP Window Image]

The solution is found by using the View menu and Solution Values instruction:
From the Solution values window the optimal solution, \( y_1 = 2 \) and \( y_2 = 2 \), can be found. The objective function having the value 10 at this (global optimal) point.

**A THIRD EXAMPLE**

The first example was a convex MINLP problem and the second one an INLP problem with one quasi-convex constraint. In order to illustrate the use of the extended quasi-convex parameters in the ECP algorithm, there follows a MINLP problem with a pseudo-convex objective function. In the main ECP window below, the problem has been defined and solved using the extended quasi-convex parameters. As can be found from the window the objective function has been defined as an objective function constraint of the form given by Eq. (1). This has not yet anything to do with the solution strategy but it illustrates the mathematical form in which the problem is defined. In order to define that the original objective function was pseudo-convex, the corresponding objective function constraint is defined by the `type=pco`. Then the algorithm understands that the corresponding constraint is an objective function constraint containing a pseudo-convex objective function and an additional variable, which then is minimized. The solution strategies for a problem with a pseudo-convex objective function as has been described in the first sections, can thereafter be used.

When using the default solution strategy, the solution to this problem is obtained within 8 iterations as can be found from the main window below. In addition to the results that can be viewed from the main window, an intermediate file `alpha.pco` gives information on how the solution was obtained. Below, a printout of the `alpha.pco` file is also given.
From the *alpha.pco* file we find from the second column that 4 different contours have been visited during the iteration procedure. Thus, in order to obtain the global optimal solution to the actual MINLP problem with a pseudo-convex objective function, in principle, 5 MINLP problems, \((P^n_r)\), with linear objective function and pseudo-convex inequality constraints have been solved within a total of 8 MILP iterations. The parameters used when solving problem prob10.ecp are given below.
In addition to the examples (prob01.ecp, prob03.ecp and prob10.ecp) illustrated in the previous chapter, a set of different MINLP problems with their solutions using the Alpha-ECP solver is given in Appendix E. All problems were solved on a 2000 MHz Pentium4 PC in less than one, or at most, in a few CPU seconds with the given version of the Alpha-ECP solver. CPLEX version 7.0 was used as the intermediate MILP solver. The problem sizes and the exact solution times when solving the problems are given in Appendix D. In addition the problem files Prob01.ecp-Prob12.ecp are given in the subdirectory: C:\Program Files\AlphaEcp\Problems. The used parameter settings are in the corresponding parameter files Prob01.par-Prob12.par and given in the same sub directory as the problem files.

**DISCUSSION**

An interactive program for solving MINLP problems by the extended cutting plane method was described in the present user’s guide. The algorithm ensures global convergence for MINLP problems with a pseudo-convex objective function and pseudo-convex constraints but the algorithm also has good convergence properties on general non-convex problems. The solutions of a set of small test examples were given in order to illustrate the use of the Alpha-ECP program and the convergence properties of the algorithm. When solving large problems the problem size (number of variables and constraints) is restricted only by the available memory size.
REFERENCES


APPENDIX A.
A Short Trouble Shooting Guide

I cannot get any solution at all when trying to solve the first example problem. Why?

1) Probably you use a regional setting such that “,” is used instead of “.” to separate the decimal part in a real number. Change your settings.
2) Probably you have not loaded the FORTRAN-90 compiler correctly. Check.
3) Probably you have not loaded the CPLEX callable library correctly. Check.
4) Probably you do not have set the path to the CPLEX callable library.
5) Probably you do not have copied a version of the CPLEX callable library to the $C:/ProgramFiles/AlphaECP$ directory as a CPLEX.DLL file.

Paths can be set by clicking on the icons My Computer, Control Panel and System. From the System Properties window you select Environment and add the path to the CPLEX callable library. A System Properties window is shown below.

The non-linear constraints in my problem are not correctly solved. Why?

1) Probably you have used a bad syntax. Check the files $alpha2.for$ and $alpha.nlc$. If the non-linear constraints are correctly written in these files then the generated FORTRAN-90 programs should be OK.
2) Check if all variables of the problem are in the file *alpha.var* (or in the common list of the *alpha2.for* program). If this is not the case, then all variables of the problem are not defined in the *LINEAR* part of the problem. Write additional linear constraints in the linear part of your problem (for example: \(c215: x_{15} \leq 100\)).

3) Probably you use a variable with the name “e”. Such a variable name is unfortunately not allowed, since the MINLP solver cannot make difference between a variable “e” and “e” in an exponential format. Change the name of the variable.

4) Probably the name of some variable contain more than 16 characters. Change the variable name.

5) If you use your own subroutine, then remember that all variables involved in the problem must appear in the linear part of the problem. If this is the case, all variable names are put into the common block in the right order in the temporary *alpha2.for* file which is generated before stopping. Use this common block in the user supplied sub program. Also remember to define the number of non-linear constraints in your sub program. (See examples in Appendix E).

6) Probably you have used the constraint type *pco*. This constraint type is only allowed for one constraint and it must be the objective function constraint, written in the form given by Eq. (1). Furthermore, observe that the rewritten objective function must contain only the additional objective function variable in this case.

**The program terminates at a clearly non-optimal point. Why ?**

1) If the problem is a MINLP problem with quasi-convex constraints, then the termination-criterion has to be the quasi-convex one. Change the parameter settings to *Quasi-Convex Settings* in the *Parameters* window. Also check if the epsilon values are sufficiently small.

2) Probably you have used the constraint type *cvx*. This type can only be used on convex constraints. If the constraint is not convex, change the type of your constraint.

3) If you have used the constraint type *pco* then you must use extended quasi-convex settings.

**The program does not seem to terminate at all. Why ?**

1) Probably the epsilon values in the termination criterions are too small. Remember also that Epsilon揭晓 should be less than Epsilon揭晓, and that the epsilon values are absolute errors. If the problem is non-convex and not even quasi-convex, then it may help to switch off the local-underestimators when defining the *ECP parameters*.

**How can I use implicit non-linear constraints ?**

1) Select *Allow Modified Fortran Files* in the *Settings* menu. Write a non-linear constraint in the *Non-linear constraints* section (if no other explicit non-linear constraints are otherwise involved in the problem) in your problem file. Once the program is started, it stops when the *alpha2.for* program has been generated. At this point the sub program *alpha2.for* (used for calculating the non-linear constraints and for calculating the derivatives) can be modified. When the implicit expression/s have been included in this file, then the procedure can be continued by pressing the *OK* button in the *Modify Fortran Files* window.

**How should non-linear equality constraints be handled ?**

1) The actual version of the program cannot handle non-linear equality constraints in
form. However, you can, for example, try to take the square of the non-linear constraint and write it as an inequality constraint. (See examples in Appendix E.)

The program seems to cut off the problem file when it is imported. Why?

1) Probably you use Windows 95. In that case the size of your problem file may be restricted to 64kbytes.

The numerical derivatives do not seem to be correctly calculated. Why?

1) Probably the analytical expressions for the derivatives are not correct. Change.
2) If the derivative at hand should be calculated numerically, then check if the step size in the forward difference approximation is appropriate. Change the step size.
3) Probably a forward difference approximation is not appropriate for your problem. Then use Allow Modified Fortran Files in the settings menu and make changes in the sub program numgrd() where the numerical derivatives are calculated. The subroutine numgrd() is in the file alpha3.for.

APPENDIX B.

Automatically Generated FORTRAN-90 Sub-Program to Calculate Non-Linear Constraints and Derivatives for the First Example Problem.

Alpha2.for

```fortran
subroutine nlcon(igecp,gecp,ngecp,nvecp,idecp)
  implicit real*8 (a-z)
  integer mgecp,mvecp,ngecp,nvecp,ngcecp,ndcecp,ircecp,maxecp
  common /ecp/mgecp,mvecp,ngcecp,ndcecp,ircecp,maxecp
  c ################### Subroutine nlcon ################################
  c A subroutine producing values on a given non-linear constraint
  c or producing values on a(n analytically) given derivative
  c #### Inputs #########################################################
  c  ngcecp = index of the non-linear constraint
  c  nvecp = index of the variable (see outputs)
  c #### Outputs if nvecp <= 0 ##########################################
  c  gecp = value of the actual non-linear constraint
  c  igecp = 1 this is a convex constraint
  c  igecp = 2 this is a quasi-convex constraint
  c  igecp = 3 this is a general non-linear constraint
  c  igecp = 4 this is a convex objective function constraint
  c  igecp = 5 this is a (not used) termination constraint
  c #### Outputs if nvecp >  0 ##########################################
  c  gecp = value of the derivative (if idecp=0 or 1)
  c  idecp = 0 the derivative value defined by default expression
  c  idecp = 1 " " calculated " analytical expression
  c  idecp = 2 " " should be calculated numerically
  c #### Variables from/to common ecp ###################################
  c  mgecp = number of non-linear constraints (t)
  c  ngcecp = sum of non-linear constraint evaluations (f/t)
  c  ndcecp = sum of analytical derivative evaluations (f/t)
  c #### If ngcecp=0 then only mgecp (number of nlc) on output ###########
  c #### TW-20030129 ####################################################
end subroutine nlcon
```
common /xtoecp/y1,y2,y3,x1,x3,u,x2
mgecp= 3
if(ngecp.le.0) return
if(nvecp.le.0) ngcecp=ngcecp+1
idecp=2

### Section for Non-linear constraint 1
#### Defined type: Convex constraint
###
If((ngecp.eq.1).and.(nvecp.le.0)) then
  igecp=0
  gecp= -0.8*0.96*log(x1-x2+1.)+0.8*x3
  elseif((ngecp.eq.1).and.(nvecp.eq.4))then
    idecp=1
    gecp= -0.96*(1.0/(x1-x2+1.))
  elseif((ngecp.eq.1).and.(nvecp.eq.7))then
    idecp=1
    gecp= -0.8*(1.0/(x2+1.))-0.96*(-1.0/(x1-x2+1.))
  elseif((ngecp.eq.1).and.(nvecp.eq.5))then
    idecp=1
    gecp=  0.8
  elseif((ngecp.eq.1).and.(nvecp.gt.0))then
    idecp=0
    gecp= 0
Endif

### Section for Non-linear constraint 2
#### Defined type: Convex constraint
###
If((ngecp.eq.2).and.(nvecp.le.0)) then
  igecp=0
  gecp= -log(x2+1.)-1.2*log(x1-x2+1.)+x3+2.0*y3-2.0
  elseif((ngecp.eq.2).and.(nvecp.eq.4))then
    idecp=1
    gecp= -1.2*(1.0/(x1-x2+1.))
  elseif((ngecp.eq.2).and.(nvecp.eq.7))then
    idecp=1
    gecp= -1.0*(1.0/(x2+1.))-1.2*(-1.0/(x1-x2+1.))
  elseif((ngecp.eq.2).and.(nvecp.eq.5))then
    idecp=1
    gecp=  1.0
  elseif((ngecp.eq.2).and.(nvecp.eq.3))then
    idecp=1
    gecp=  2.0
  elseif((ngecp.eq.2).and.(nvecp.gt.0))then
    idecp=0
    gecp= 0
Endif

### Section for Non-linear constraint 3
#### Defined type: Convex constraint
###
If((ngecp.eq.3).and.(nvecp.le.0)) then
  igecp=0
  gecp= -18.0*19.2*log(x1-x2+1.)+10.0-u
  elseif((ngecp.eq.3).and.(nvecp.eq.4))then
    idecp=1
    gecp= -19.2*(1.0/(x1-x2+1.))
  elseif((ngecp.eq.3).and.(nvecp.eq.7))then
    idecp=1
    gecp= -18.0*(1.0/(x2+1.))-19.2*(-1.0/(x1-x2+1.))
  elseif((ngecp.eq.3).and.(nvecp.eq.6))then


APPENDIX C.

CONTENT OF SOME DATA FILES GENERATED BY THE Alpha-ECP SOLVER

Alpha0.dat
Name of the problem file in extended LP format
Name of the generated LP file to CPLEX
k, r, t, iflue, ifleps, iflend, iflmip, iflalp, iflrel, iflue, iflech, ifldfi,
beta, gamma, epsg, epsh, delta, alphamax, gmax, ngmax, ifldm1, ifldm2

Alpha1.dat
Zmin, z(1), z(2), z(3), …., z(nv)

Alpha2.dat
i, k, r, Alpha, g_k, c_k, irecsta, irecsto

Alpha3.dat
nv1 dg(nv1)
nv2 dg(nv2)

Alpha4.dat
i, k, r, Alpha, l_j(z_k), g_i(z_k) (l_j > g_i)

Alpha.cmd
fl32 /4R8 /LD /G5 alpha.for alpha1.obj alpha2.for alpha3.for

Alpha.cpu
k, r, t, CPU_milp, CPU_ecp, CPU_total, iflmip

Alpha.ecp
MINLP problem in the main ECP window.

Alpha.int
k, r, t, Zmin, Alphamax, gmax, i, number of constraint evaluations, number of derivative evaluations

Alpha.lp
Actual MILP problem transferred to the MILP solver.

Alpha.nl
Non-linear constraints

Alpha.val
Zmin, z(1), z(2), z(3), …., z(nv)

Alpha.var
List of variable names

Alpha.vda
Type, variable name

Alpha.pco
Iter, pcomin, fpco, pcomin-mu, sol-type

Alpha1.zzz
k,ista,isto,fpco

Alpha2.zzz
nv1 z(nv1)
nv2 z(nv2)
APPENDIX D.

Problem sizes as well as solution times when solving the problems Prob01-Prob12 with Alpha-ECP version 5.101 (using CPLEX version 7.0 as the MILP solver) on a 2000 MHz Pentium4 PC.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Number of Non-linear Constraints</th>
<th>Number of Linear Constraints</th>
<th>Objective function type</th>
<th>Constraints In total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prob01</td>
<td>MINLP</td>
<td>3</td>
<td>3 Convex</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Prob02</td>
<td>INLP</td>
<td>5</td>
<td>5 Linear</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Prob03</td>
<td>INLP</td>
<td>1</td>
<td>1 Linear</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Prob04</td>
<td>MINLP</td>
<td>6</td>
<td>6 Linear</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Prob05</td>
<td>NLP</td>
<td>2</td>
<td>2 Linear</td>
<td>2 (4)</td>
<td></td>
</tr>
<tr>
<td>Prob06</td>
<td>NLP</td>
<td>2</td>
<td>2 Linear</td>
<td>2 (4)</td>
<td></td>
</tr>
<tr>
<td>Prob07</td>
<td>NLP</td>
<td>24</td>
<td>24 Non-convex</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Prob08</td>
<td>MINLP</td>
<td>1</td>
<td>1 Pseudo-Convex *</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Prob09</td>
<td>NLP</td>
<td>1</td>
<td>1 Non-convex</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Prob10</td>
<td>MINLP</td>
<td>1</td>
<td>1 Pseudo-convex</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Prob11</td>
<td>MINLP</td>
<td>1</td>
<td>1 Pseudo-convex</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>Prob12</td>
<td>MINLP</td>
<td>14</td>
<td>14 Linear</td>
<td>213</td>
<td></td>
</tr>
</tbody>
</table>

• The objective function is convex but defined as a pseudo-convex objective function constraint in this problem.

Table 1. Number of constraints in the problems Prob01-Prob12.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>Number of Variables</th>
<th>Number of Non-linear Constraints</th>
<th>Number of Linear Constraints</th>
<th>Objective function type</th>
<th>Constraints In total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prob01</td>
<td>MINLP</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob02</td>
<td>INLP</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob03</td>
<td>INLP</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob04</td>
<td>MINLP</td>
<td>27</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob05</td>
<td>NLP</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob06</td>
<td>NLP</td>
<td>15</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob07</td>
<td>NLP</td>
<td>30</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob08</td>
<td>MINLP</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob09</td>
<td>NLP</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob10</td>
<td>MINLP</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob11</td>
<td>MINLP</td>
<td>233</td>
<td>233</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prob12</td>
<td>MINLP</td>
<td>114</td>
<td>114</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Number of variables in the problems Prob01-Prob12.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Type</th>
<th>ECP</th>
<th>CPU-time (s)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Iterations</td>
<td>CPU-alpha</td>
<td>CPU-MILP</td>
</tr>
<tr>
<td>Prob01</td>
<td>MINLP</td>
<td>11</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Prob02</td>
<td>INLP</td>
<td>25</td>
<td>0.03</td>
<td>0.07</td>
</tr>
<tr>
<td>Prob03</td>
<td>INLP</td>
<td>10</td>
<td>0.05</td>
<td>0.01</td>
</tr>
<tr>
<td>Prob04</td>
<td>MINLP</td>
<td>69</td>
<td>0.13</td>
<td>0.33</td>
</tr>
<tr>
<td>Prob05</td>
<td>NLP</td>
<td>27</td>
<td>0.06</td>
<td>0.01</td>
</tr>
<tr>
<td>Prob06</td>
<td>NLP</td>
<td>21</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Prob07</td>
<td>NLP</td>
<td>80</td>
<td>0.18</td>
<td>0.14</td>
</tr>
<tr>
<td>Prob08</td>
<td>MINLP</td>
<td>81</td>
<td>0.32</td>
<td>2.16</td>
</tr>
<tr>
<td>Prob09</td>
<td>NLP</td>
<td>29</td>
<td>0.10</td>
<td>0.01</td>
</tr>
<tr>
<td>Prob10</td>
<td>MINLP</td>
<td>8</td>
<td>0.05</td>
<td>0.01</td>
</tr>
<tr>
<td>Prob11</td>
<td>MINLP</td>
<td>129</td>
<td>2.21</td>
<td>26.44</td>
</tr>
<tr>
<td>Prob12</td>
<td>MINLP</td>
<td>23</td>
<td>0.03</td>
<td>128.11</td>
</tr>
</tbody>
</table>

Table 3. ECP-iterations and CPU-times when solving problems Prob01-Prob12.
APPENDIX E.
Prob02.ecp

Quasi-Convex Trim-Loss Problem
Computers chem Engng., 22, 357-365.
### Active Parameter Values

**ECP-Parameters:**
- ECP Linearization Strategy = 'Add Linearization for Max. g'
- Starting with problem = MIP
- Starting point = 'Automatically determined'
- Alpha = 'Updating, RHS only'
- Beta = 1.3
- Gamma = 5
- Local Underestimator = 'Not used'
- Feasible Underestimator = 'alpha > gk/epsh when gmax < epsg'
- Line Search not used
- ECP Iteration Limit = 100
- ECP Termination Criteria = 'Quasi-Convex MINLP'
- Epsilon_g = 0.001
- Epsilon_z = 0.1
- Epsilon_h strategy = Epsilon_h = Epsilon_z * Norm of gradient at h
- Epsilon_I not used

**Cplex-Parameters:**
- Branching Direction = 'Automatically determined'
- Preprocessing presolve = 1
- Node selection strategy = 'Best-bound search'
- Subproblem LP algorithm = 'Dual simplex'
- Variable selection strategy = 'Branch variable automatically selected'
- Mip strategy backtrack = 1
- Actual Mip solutions limit = 500000
- Tree memory limit = 128
- Relative mipgap tolerance = 0.0001
- Integrality tolerance = 0.00001

**Other:**
- Delta = 0.001
**Prob04.ecp**

**Synthesis Problem**

### Active Parameter Values

<table>
<thead>
<tr>
<th>ECP-Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECP Linearization Strategy = &quot;Add Linearization for Max. g&quot;</td>
</tr>
<tr>
<td>Starting with problem = MIP</td>
</tr>
<tr>
<td>Starting point = &quot;Automatically determined&quot;</td>
</tr>
<tr>
<td>Alpha = &quot;Updating, RHS only&quot;</td>
</tr>
<tr>
<td>Beta = 1.3</td>
</tr>
<tr>
<td>Gamma = 2</td>
</tr>
<tr>
<td>Local Underestimator = &quot;Not used&quot;</td>
</tr>
<tr>
<td>Feasible Underestimator = &quot;alpha &gt;= g / epsilon when g &lt;= epsilon&quot;</td>
</tr>
<tr>
<td>Line Search not used</td>
</tr>
<tr>
<td>ECP Iteration Limit = 100</td>
</tr>
<tr>
<td>ECP Termination Criteria = &quot;Quasi-Convex MINLP&quot;</td>
</tr>
<tr>
<td>Epsilon_g = 0.001</td>
</tr>
<tr>
<td>Epsilon_z = 0.001</td>
</tr>
<tr>
<td>Epsilon_h strategy = Epsilon_h = Epsilon_z &quot;Norm of gradient at k&quot;</td>
</tr>
<tr>
<td>Epsilon_f not used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPLEX-Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching Direction = &quot;Automatically determined&quot;</td>
</tr>
<tr>
<td>Preprocessing presolve = 1</td>
</tr>
<tr>
<td>Node selection strategy = &quot;Best bound search&quot;</td>
</tr>
<tr>
<td>Subproblem LP algorithm = &quot;Dual simplex&quot;</td>
</tr>
<tr>
<td>Variable selection strategy = &quot;Branch variable automatically selected&quot;</td>
</tr>
<tr>
<td>Mip strategy backtrack = 1</td>
</tr>
<tr>
<td>Actual Mip solutions limit = 500000</td>
</tr>
<tr>
<td>Tree memory limit = 128</td>
</tr>
<tr>
<td>Relative mipgap tolerance = 0.0001</td>
</tr>
<tr>
<td>Integrality tolerance = 0.00001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta = 0.001</td>
</tr>
</tbody>
</table>

[Print... Close]
**Prob05.ecp**

*Global Optimization Problem*

## Active Parameter Values

**ECP-Parameters:**
- **ECP Linearization Strategy** = 'Add Linearization for Max. g'
- **Starting point** = 'Automatically determined'
- **Alpha** = 'Updating, RHS only'
- **Beta** = 1.3
- **Gamma** = 2
- **Local Underestimator** = 'Not used'
- **Feasible Underestimator** = 'alpha > gk/epsilon when gmax < epsilon'
- **Line Search** = not used
- **ECP Iteration Limit** = 100
- **ECP Termination Criteria** = 'Quasi-Convex MINLP'
- **Epsilon_g** = 0.001
- **Epsilon_z** = 0.001
- **Epsilon_h** strategy = $\Sigma h$.h = $\Sigma h$.z $^*$ Norm of gradient at k
- **Epsilon_I** = not used

**CPLEX-Parameters:**
- **Branching Direction** = 'Automatically determined'
- **Preprocessing** = presolve = 1
- **Node selection strategy** = 'Best-bound search'
- **Subproblem LP algorithm** = 'Dual simplex'
- **Variable selection strategy** = 'Branch variable automatically selected'
- **Mip strategy backtrack** = 1
- **Actual Mip solutions limit** = 2100000000
- **Trees memory limit** = 128
- **Relative mipgap tolerance** = 0.0001
- **Integrity tolerance** = 0.00001

**Other:**
- **Delta** = 0.001

[Print... Close]
Prob06.ecp

Global Optimization Problem

### Active Parameter Values

**ECP-Parameters:**

- **ECP Linearization Strategy:** 'Add Linearization for Max. g'
- **Starting with problem:** MIP
- **Starting point:** Automatically determined
- **Alpha:** 'Updating, RHS only'
- **Beta:** -1.3
- **Gamma:** 2
- **Local Underestimator:** 'Not used'
- **Feasible Underestimator:** 'alpha >= gk/epsin when gmax < epsg'
- **Line Search not used**
- **ECP Iteration Limit:** 100
- **ECP Termination Criteria:** 'Quasi-Convex MINLP'
- **Epsilon_g:** 0.001
- **Epsilon_z:** 0.001
- **Epsilon_h strategy:** Epsilon_h = Epsilon_z * Norm of gradient at k
- **Epsilon_f:** not used

**CPLEX-Parameters:**

- **Branching Direction:** 'Automatically determined'
- **Preprocessing presolve:** 1
- **Node selection strategy:** 'Best-bound search'
- **Subproblem LP algorithm:** 'Dual simplex'
- **Variable selection strategy:** 'Branch variable automatically selected'
- **Mip strategy backtracking:** 1
- **Actual Mip solutions limit:** 500000
- **Tree memory limit:** 128
- **Relative mipgap tolerance:** 0.0001
- **Integrity tolerance:** 0.000001

**Other:**

- **Delta:** 0.001
Prob07.ecp
Design Application
Engineering Optimization, Methods and Applications, John Wiley and Sons. Pages 421-430.

The ECP-problem file

Minimize Z
Subject to
c1: V1 - 1.2 B1 >= 0
c2: V1 - 1.5 B2 >= 0
c3: V1 - 1.1 B3 >= 0
c4: V2 - 1.4 B1 >= 0
c5: V2 - 1.2 B3 >= 0
c6: V3 - 1.0 B1 >= 0
c7: V3 - 1.2 B2 >= 0
c8: V3 - 1.0 B3 >= 0
c9: R2 - R3 <= 0
c10: R4 - R5 <= 0
c11: R2 - R5 <= 0
c12: R3 - R2 = 0
c13: T1 <= 6
c14: T2 <= 6
c15: T3 <= 6
Non-linear constraints
s1: 962(V1)^0.56 + 562(V2)^0.29 + 1205(V3)^0.52 <= 280(V1)^0.22 + 230(V2)^0.44 + 210(V3)^0.52
250(V4)^0.40 + 100(V5)^0.83 <= 6
c22: T2 >= 5
c23: T3 >= 5

Non-linear constraints

g1: 592*(V1**0.65)+582*(V2**0.39)+1200*(V3**0.52)+
370*(R1**0.22)+250*(R2**0.40)+210*(R3**0.62)+
250*(R4**0.40)+200*(R5**0.83) - Z <= 0
Default: 0
dg1/dV1: 0.65*592/V1**0.35
dg1/dV2: 0.39*582/V2**0.61
dg1/dV3: 0.52*1200/V3**0.48
dg1/dR1: 0.22*370/R1**0.78
dg1/dR2: 0.40*250/R2**0.60
dg1/dR3: 0.62*210/R3**0.38
dg1/dR4: 0.40*250/R4**0.60
dg1/dR5: 0.83*200/R5**0.17
dg1/dZ: -1
g2: 400000/B1*T1+300000/B2*T2+100000/B3*T3-8000 <= 0
Default: 0
dg2/dB1: -400000*T1/B1/B1
dg2/dB3: -100000*T3/B3/B3
dg2/dT1: 400000/B1
dg2/dT2: 300000/B2
g2/dT3: 100000/B3
g3: 1.2*B1/R1-T1 <= 0
Default: 0
dg3/dB1: 1.2/R1
dg3/dR1: -1.2*B1/R1/R1
dg3/dT1: -1
g4: 1.2*B1/R2-T1 <= 0
Default: 0
dg4/dB1: 1.2/R2
dg4/dR2: -1.2*B1/R2/R2
dg4/dT1: -1
g5: 1.2*B1/R3-T1 <= 0
Default: 0
dg5/dB1: 1.2/R3
dg5/dR3: -1.2*B1/R3/R3
dg5/dT1: -1
g6: 1.4*B1/R4-T1 <= 0
Default: 0
dg6/dB1: 1.4/R4
dg6/dR4: -1.4*B1/R4/R4
dg6/dT1: -1
g7: 1.4*B1/R5-T1 <= 0
Default: 0
dg7/dB1: 1.4/R5
dg7/dR5: -1.4*B1/R5/R5
dg7/dT1: -1
g8: 1.5*B2/R1-T2 <= 0
Default: 0
dg8/dB2: 1.5/R1
dg8/dR1: -1.5*B2/R1/R1
dg8/dT2: -1
g9: 1.5*B2/R2-T2 <= 0
Default: 0
dg9/dB2: 1.5/R2
dg9/dR2: -1.5*B2/R2/R2
dg9/dT2: -1
g10: 1.5*B2/R3-T2 <= 0
Default: 0
dg10/dB2: 1.5/R3
dg10/dR3: -1.5*B2/R3/R3
dg10/dT2: -1
g11: 1.5*B2/R5-T2 <= 0
Default: 0
dg11/dB2: 1.5/R5
dg11/dR5: -1.5*B2/R5/R5
dg11/dT2: -1
g12: 1.1*B3/R1-T3 <= 0
Default: 0
dg12/dB3: 1.1/R1
dg12/dR1: -1.1*B3/R1/R1
dg12/dT3: -1
g13: 1.1*B3/R2-T3 <= 0
Default: 0
dg13/dB3: 1.1/R2
dg13/dR2: -1.1*B3/R2/R2
dg13/dT3: -1
g14: 1.1*B3/R3-T3 <= 0
Default: 0
dg14/dB3: 1.1/R3
dg14/dR3: -1.1*B3/R3/R3
dg14/dT3: -1
g15: 1.2*B3/R4-T3 <= 0
Default: 0
dg15/dB3: 1.2/R4
dg15/dR4: -1.2*B3/R4/R4
dg15/dT3: -1
g16: 1.2*B3/R5-T3 <= 0
Default: 0
dg16/dB3: 1.2/R5
dg16/dR5: -1.2*B3/R5/R5
dg16/dT3: -1
g17: 1.2*B1/R1 + 3 + 1.2*B1/R2 - T1 <= 0
Default: 0
dg17/dB1: 1.2/R1+1.2/R2
dg17/dR1: -1.2*B1/R1/R1
dg17/dR2: -1.2*B1/R2/R2
dg17/dT1: -1
g18: 1.2*B1/R3 + 1 + 1.4*B1/R4 - T1 <= 0
Default: 0
dg18/dB1: 1.2/R3+1.4/R4
dg18/dR3: -1.2*B1/R3/R3
dg18/dR4: -1.4*B1/R4/R4
dg18/dT1: -1
g19: 1.4*B1/R5 + 4 - T1 <= 0
Default: 0
dg19/dB1: 1.4/R5
dg19/dR5: -1.4*B1/R5/R5
dg19/dT1: -1
g20: 1.5*B2/R1 + 6 + 1.5*B2/R2 - T2 <= 0
Default: 0
dg20/dB2: 1.5/R1+1.5/R2
dg20/dR1: -1.5*B2/R1/R1
dg20/dR2: -1.5*B2/R2/R2
dg20/dT2: -1
g21: 1.5*B2/R5 + 8 - T2 <= 0
Default: 0
dg21/dB2: 1.5/R5
dg21/dR5: -1.5*B2/R5/R5
dg21/dT2: -1
g22: 1.1*B3/R1 + 2 + 1.1*B3/R2 - T3 <= 0
Default: 0
dg22/dB3: 1.1/R1 + 1.1/R2
dg22/dR1: -1.1*B3/R1/R1
dg22/dR2: -1.1*B3/R2/R2
dg22/dT3: -1

g23: 1.1*B3/R3 + 2 + 1.2*B3/R4 - T3 <= 0
Default: 0
dg23/dB3: 1.1/R3 + 1.2/R4
dg23/dR3: -1.1*B3/R3/R3
dg23/dR4: -1.2*B3/R4/R4
dg23/dT3: -1

g24: 1.2*B3/R5 + 4 - T3 <= 0
Default: 0
dg24/dB3: 1.2/R5
dg24/dR5: -1.2*B3/R5/R5
dg24/dT3: -1

Solution values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>155153.5436575687</td>
</tr>
<tr>
<td>V1</td>
<td>1100.53962191846</td>
</tr>
<tr>
<td>B1</td>
<td>913.955157873337</td>
</tr>
<tr>
<td>B2</td>
<td>733.692414545642</td>
</tr>
<tr>
<td>B3</td>
<td>913.955157873337</td>
</tr>
<tr>
<td>V2</td>
<td>1279.53722102267</td>
</tr>
<tr>
<td>V3</td>
<td>913.955157873337</td>
</tr>
<tr>
<td>R2</td>
<td>365.579732331263</td>
</tr>
<tr>
<td>R3</td>
<td>365.579732331263</td>
</tr>
<tr>
<td>R4</td>
<td>469.6258373931453</td>
</tr>
<tr>
<td>R5</td>
<td>455.6258373931453</td>
</tr>
<tr>
<td>R1</td>
<td>1399.1632390451</td>
</tr>
<tr>
<td>T1</td>
<td>6.70396433934926</td>
</tr>
<tr>
<td>T2</td>
<td>10.3894267125785</td>
</tr>
<tr>
<td>T3</td>
<td>7.13617532404846</td>
</tr>
</tbody>
</table>
### Active Parameter Values

**ECP-Parameters:**
- ECP Linearization Strategy = 'Add Linearization for Max. g'
- Starting with problem = 'MIP'
- Starting point = 'Automatically determined'
- Alpha = 'Updating, RHS only'
- Beta = 1.3
- Gamma = 2
- Local Underestimator = 'Not used'
- Feasible Underestimator = 'alpha >= gk/epsk when gmax < epsg'
- Line Search not used
- ECP Iteration Limit = 200
- ECP Termination Criteria = 'Quasi-Convex MINLP'
- Epsilon_g = 0.001
- Epsilon_z = 0.1
- Epsilon_h strategy = Epsilon_h = Epsilon_z * Norm of gradient at k
- Epsilon_f not used

**Cplex-Parameters:**
- Branching direction = 'Automatically determined'
- Preprocessing presolve = 1
- Node selection strategy = 'Best bound search'
- Subproblem LP algorithm = 'Dual simplex'
- Variable selection strategy = 'Branch variable automatically selected'
- MIP strategy backtrack = 1
- Actual MIP solutions limit = 500000
- Tree memory limit = 128
- Relative mipgap tolerance = 0.0001
- Integrality tolerance = 0.00001

**Other:**
- Delta = 0.001
**Prob08.ecp**

*Fourier Transform Infrared Spectroscopy Problem*


---

**Modified FORTRAN-90 file for the non-linear constraints**

```fortran
c ###########################################################################
c c Subroutine producing values on a given non-linear constraint c or producing values on a(n analytically) given derivative c ###########################################################################
c c Inputs: c ngecp = index of the non-linear constraint c nvecp = index of the variable (see outputs) c c Outputs if nvecp <= 0 c gecp = value of the actual non-linear constraint c igecp = 0 this is a convex constraint c igecp = 1 this is a quasi-convex constraint c igecp = 2 this is a pseudo-convex objective function constraint c igecp = 3 this is a general non-linear constraint c c Outputs if nvecp > 0 c gecp = value of the derivative (if idecp=0 or 1) c idecp = 0 the derivative value defined by default expression c idecp = 1 " " calculated " analytical expression c idecp = 2 " " should be calculated numerically c c Variables from/to common ecp c mgecp = number of non-linear constraints (t) c ngcecp = sum of non-linear constraint evaluations (f/t) c ndcecp = sum of analytical derivative evaluations (f/t) c c If ngcecp=0 then only mgecp (number of nlc) on output c c TW/1999-03-31 FTIR Problem c c ###########################################################################
```
subroutine nlcon(igecp,gecp,ngecp,nvecp,idecp)
implicit real*8 (a-h,o-z)
dimension c(3,8),a(10,8),e(3),ri(3,3),eTri(3)
integer mgecp,mvecp,ngcecp,ndcecp,
1 igecp,idecp,ircecp,maxecp
common /ecp/mgecp,mvecp,ngcecp,ndcecp,ircecp,maxecp
common /xtoecp /
1 ,y11,y12,y13,y14,y15,y16,y17,y18,y19,y10
1 ,y21,y22,y23,y24,y25,y26,y27,y28,y29,y210
1 ,y31,y32,y33,y34,y35,y36,y37,y38,y39,y310,pT(10,3)
common /internal/dg(30)
data ri/1,0,0, 0,1,0, 0,0,1/e/0,0,0/
data c/
1  5.02, 0.97, 0.00,
1  2.04, 3.51, 2.20,
1  3.53, 3.51, 0.80,
1  7.02, 3.51, 0.00,
1  0.00, 7.00, 1.40,
1  1.00, 0.00, 2.20,
1  1.00, 2.01, 1.40,
1  2.04, 0.97, 0.80/
data a/
1   .0003,.0007,.0066,.0044,.0208,.0518,.0036,.0507,.0905,.0016,
1   .0764,.0003,.0789,.0186,.0605,.1656,.0035,.0361,.0600,.0209,
1   .0318,.0004,.0275,.0180,.0601,.1491,.0032,.0433,.0754,.0636,
1   .0007,.0009,.0043,.0179,.0604,.1385,.0051,.0635,.1098,.0010,
1   .0534,.0005,.0704,.0351,.0981, 2389,.0015,.0048,.0038,.0132,
1   .0773,.0008,.0683,.0024,.0025,.0248,.0094,.0891,.1443,.0203,
1   .0536,.0005,.0842,.0108,.0394,.1122,.0015,.0213,.0420,.0139,
1   .0320,.0003,.0309,.0052,.0221,.0633,.0024,.0310,.0574,.0057/
mgecp=1
if(ngecp.le.0) return
if(nvecp.le.0) ngcecp=ngcecp+1
if(nvecp.le.0) ndcecp=ndcecp+1
idecp=1
if(nvecp.eq.1)gecp=-1.
if((nvecp.ge.2).and.(nvecp.le.31))gecp=2.
if((nvecp.ge.32).and.(nvecp.le.61))gecp=dg(nvecp-31)
if(nvecp.gt.0) return

ggecp=1
igecp=2
ncom=3
nabs=10
nexp=8
dgd=1,ncom*nabs
end do
end do
end do
end do
ndcecp=ndcecp+1

## Section for Non-linear constraint

### Defined type: Pseudo-convex objective function constraint
### Linear objective function variable = z

ngecp=1
igecp=2
ncom=3
nabs=10
nexp=8
dgecp=-z + 2*y11+ 2*y12+ 2*y13+ 2*y14+ 2*y15 +
1 2*y16+ 2*y17+ 2*y18+ 2*y19+ 2*y10 +
1 2*y21+ 2*y22+ 2*y23+ 2*y24+ 2*y25 +
1 2*y26+ 2*y27+ 2*y28+ 2*y29+ 2*y30 +
1 2*y31+ 2*y32+ 2*y33+ 2*y34+ 2*y35 +
1 2*y36+ 2*y37+ 2*y38+ 2*y39+ 2*y40
do idg=1,ncom*nabs
dg(idg)=0.
end do
end do
end do
end do
end do
end do
end do
end do
do ic=1,ncom
  eTri(ic)=0.
  do jc=1,ncom
    eTri(ic)=eTri(ic)+e(jc)*ri(jc,ic)
  end do
end do

##### The derivatives with respect to all parameters can be calculated at the same time the non-linear constraint is calculated in this example. The values are stored in the vector dg and can be picked up from there when needed (the lines in the beginning). ###

do ic=1,ncom
gecp=gecp+eTri(ic)*e(ic)
do ia=1,nabs
  idg=ia+(ic-1)*10
dg(idg)=dg(idg)-2.0*eTri(ic)*a(ia,ie)
end do
end do
return
end

### Solution values

<table>
<thead>
<tr>
<th>Objective value</th>
<th>13.9750066512528</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut Off Value</td>
<td>1E-10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>z</td>
<td>13.9711367468954</td>
</tr>
<tr>
<td>y1 8</td>
<td>1</td>
</tr>
<tr>
<td>y24</td>
<td>1</td>
</tr>
<tr>
<td>y3l</td>
<td>1</td>
</tr>
<tr>
<td>p18</td>
<td>100.09337017236</td>
</tr>
<tr>
<td>p24</td>
<td>195.136573721419</td>
</tr>
<tr>
<td>p3l</td>
<td>27.5548285737121</td>
</tr>
</tbody>
</table>
### Active Parameter Values

**ECP-Parameters:**

- **ECP Linearization Strategy:** 'Add Linearization for Max. q'
- **Starting with problem:** 'MIP'
- **Starting point:** 'Automatically determined'
- **Alpha:** 'Updating, RHS only'
- **Beta = 1.0'
- **Gamma = 2'
- **Local Underestimator:** 'Not used'
- **Feasible Underestimator:** 'alpha >= gk/eps when gmax < epsg'
- **PCD Strategy:** 'Remove and Line Search to Contour'
- **ECP Iteration Limit:** 100
- **ECP Termination Criteria:** 'IQ-C MINLP with PCD'
- **Epsilon_g = 0.001'
- **Epsilon_z = 0.001'
- **Epsilon_h strategy = Epsilon_h = Epsilon_z * Norm of gradient at h'
- **Epsilon_l = 0.1'

**Cplex-Parameters:**

- **Braching Direction:** 'Automatically determined'
- **Preprocessing presolve = 1'
- **Node selection strategy:** 'Best-bound search'
- **Subproblem LP algorithm:** 'Dual simplex'
- **Variable selection strategy:** 'Branch variable automatically selected'
- **Mip strategy backtrack = 0.65'
- **Actual Mip solutions limit = 500000'
- **Tree memory limit = 128'
- **Relative mfgap tolerance = 0.0001'
- **Integrity tolerance = 0.00001'

**Other:**

- **Delta = 0.001'
Prob09.ecp

Rosenbrock’s function

### Active Parameter Values

**ECP-Parameters:**

- **ECP Linearization Strategy** = 'Add Linearization for Max. Of'
- **Starting with problem** = 'MIP'
- **Starting point** = 'Automatically determined'
- **Alpha** = 'Updating, RHS only'
- **Beta** = -1.3
- **Gamma** = 2
- **Local Underestimator** = 'Not used'
- **Feasible Underestimator** = 'alpha > gk/epsh when gmax < epsg'
- **PCU Strategy** = 'Remove and Line Search to Contour'
- **ECP Iteration Limit** = 100
- **ECP Termination Criteria** = 'Q-C MINLP with PCD'
- **Epsilon_g** = 0.001
- **Epsilon_z** = 0.001
- **Epsilon_h strategy** = Epsilon_h = Epsilon_z * Norm of gradient at k'
- **Epsilon_l** = 0.1

**Cplex-Parameters:**

- **Branching Direction** = 'Automatically determined'
- **Preprocessing presolve** = 1
- **Node selection strategy** = 'Best-bound search'
- **Subproblem LP algorithm** = 'Dual simplex'
- **Variable selection strategy** = 'Branch variable automatically selected'
- **Mip strategy backtrack** = 1
- **Actual Mip solutions limit** = 500000
- **Trees memory limit** = 128
- **Relative mtotal tolerance** = 0.0001
- **Integrity tolerance** = 0.00001

**Other:**

- **Delta** = 0.001
Prob10.ecp
Pseudo-convex Example Problem
**Active Parameter Values**

**ECP-Parameters:**
- ECP Linearization Strategy = 'Add Linearization for Max g'
- Starting with problem = 'MIP'
- Starting point = 'Automatically determined'
- Alpha = 'Updating, RHS only'
- Beta = 1.3
- Gamma = 2
- Local Underestimator = 'Not used'
- Feasible Underestimator = 'alpha >= gk/epsin when gmax < epsg'
- PCO Strategy = Remove and Line Search to Contour
- ECP Iteration Limit = 100
- ECP Termination Criteria = 'Q-C MINLP with PCD'
- Epsilon_g = 0.0000001
- Epsilon_z = 0.001
- Epsilon_h strategy = Epsilon_h = Epsilon_z "Norm of gradient at h"  Epsilon_l = 0.001

**CPLEX-Parameters:**
- Branching Direction = 'Automatically determined'
- Preprocessing presolve = 1
- Node selection strategy = 'Best-bound search'
- Subproblem LP algorithm = 'Dual simplex'
- Variable selection strategy = 'Branch variable automatically selected'
- Mip strategy backtrack = 0.85
- Actual Mip solutions limit = 50000
- Trees memory limit = 128
- Relative mcpgap tolerance = 0.0001
- Integrality tolerance = 0.00001

**Other:**
- Delta = 0.001

[Print... Close]
Prob11.ecp

Cyclic Scheduling Problem

<table>
<thead>
<tr>
<th><strong>Active Parameter Values</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ECP-Parameters:</strong></td>
</tr>
<tr>
<td>ECP Linearization Strategy = 'Add Linearization for Max q'</td>
</tr>
<tr>
<td>Starting with problem = 'MIP'</td>
</tr>
<tr>
<td>Starting point = 'Automatically determined'</td>
</tr>
<tr>
<td>Alpha = 'Updating, RHS only'</td>
</tr>
<tr>
<td>Beta = 1.0</td>
</tr>
<tr>
<td>Gamma = 2</td>
</tr>
<tr>
<td>Local Underestimator = 'Not used'</td>
</tr>
<tr>
<td>Feasible Underestimator = 'alpha &gt; = gk/eps in when gmax &lt; epsg'</td>
</tr>
<tr>
<td>PCO Strategy = Remove and Line Search to Contour</td>
</tr>
<tr>
<td>ECP Iteration Limit = 500</td>
</tr>
<tr>
<td>ECP Termination Criteria = 'Q-C MINLP with PCO'</td>
</tr>
<tr>
<td>Epsilon_g = 0.1</td>
</tr>
<tr>
<td>Epsilon_z = 0.001</td>
</tr>
<tr>
<td>Epsilon_h strategy = Epsilon_h = Epsilon_z = Norm of gradient at k</td>
</tr>
<tr>
<td>Epsilon_l = 0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CPLEX-Parameters:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching Direction = 'Automatically determined'</td>
</tr>
<tr>
<td>Preprocessing presolve = 1</td>
</tr>
<tr>
<td>Node selection strategy = 'Best-bound search'</td>
</tr>
<tr>
<td>Subproblem LP algorithm = 'Dual simplex'</td>
</tr>
<tr>
<td>Variable selection strategy = 'Branch variable automatically selected'</td>
</tr>
<tr>
<td>Mip strategy backtrack = 0.01</td>
</tr>
<tr>
<td>Actual Mip solutions limit = 2100000000</td>
</tr>
<tr>
<td>Tree memory limit = 128</td>
</tr>
<tr>
<td>Relative mipgap tolerance = 0.0001</td>
</tr>
<tr>
<td>Integrality tolerance = 0.000001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Other:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta = 0.001</td>
</tr>
</tbody>
</table>

[Print... Close]
**Prob12.ecp**

*Facility Layout Problem*

Active Parameter Values

**ECP-Parameters:**
- ECP Linearization Strategy = Add Linearization for g ∩ Epsilon_g
- Starting with problem = MIP
- Starting point = Automatically determined
- Alpha = Constant = 1
- Beta not used
- Gamma not used
- Local Underestimator not used
- Feasible Underestimator not used
- Line Search not used
- ECP Iteration Limit = 100
- ECP Termination Criteria = 'Convex'
- Epsilon_u = 0.01
- Epsilon_l not used
- Epsilon_u not used

**CPLEX-Parameters:**
- Branching Direction = Automatically determined
- Preprocessing presolve = 1
- Node selection strategy = Best bound search
- Subproblem LP algorithm = Dual simplex
- Variable selection strategy = Branch variable automatically selected
- Mip strategy backtrack = 0.01
- Mip strategy look limit = 4
- Tree memory limit = 1024
- Relative mipgap tolerance = 0.0001
- Integrality tolerance = 0.00001

Other:
- Delta = 0.001