AIMMS Language Reference - Advanced Methods for Nonlinear Programs

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com.

AIMMS B.V.
Diakenhuisweg 29-35
2033 AP Haarlem
The Netherlands
Tel.: +31 23 5511512

AIMMS Inc.
11711 SE 8th Street
Suite 303
Bellevue, WA 98005
USA
Tel.: +1 425 458 4024

AIMMS Pte. Ltd.
55 Market Street #10-00
Singapore 048941
Tel.: +65 6521 2827

AIMMS
SOHO Fuxing Plaza No.388
Building D-71, Level 3
Madang Road, Huangpu District
Shanghai 200025
China
Tel.: ++86 21 5309 8733

Email: info@aimms.com
WWW: www.aimms.com

Chapter 17

Advanced Methods for Nonlinear Programs

Problems of nonlinear programs

For non-convex nonlinear mathematical programs (NLPs), nonlinear solvers have no guarantee of returning the global optimum. Due to the local search algorithms employed by nonlinear solvers, their solution process depends on the starting point provided by the user. Nonlinear solvers can, therefore, easily end up in, non-unique, local optima, or, even worse, may not even find a feasible solution for a given starting point.

How to counteract?

To counteract these facts, a number of possible actions can be taken.

- Use a global solver, such as BARON, to solve the NLP. Global solvers, however, usually only work well on relatively small NLP problems.
- Use a multistart algorithm to solve the NLP problem for multiple starting points in order to have a better chance to find the global optimum.
- Use the AIMMS Presolver to reduce the problem size and tighten the bounds of the remaining variables and constraints of the NLP. This will reduce the space which the nonlinear solver needs to search in order to find an optimal solution.

This chapter

This chapter discusses the presolve techniques for nonlinear programs available in AIMMS. The chapter also discusses the multistart algorithm built into AIMMS. Using the multistart algorithm will increase the total solution time, but, in general, will also improve the solution found by nonlinear solvers.

17.1 The AIMMS Presolver

The need for a presolver

Of all nonlinear solvers in AIMMS only a couple use (limited) preprocessing techniques. Therefore, AIMMS itself has implemented a presolve algorithm with the goal to reduce the size of the problem and to tighten the variable bounds, which may help the solver to solve nonlinear problems faster. Besides the BARON global solver, all nonlinear solvers in AIMMS are local solvers, i.e. the solution found by the solver is a local solution and cannot be guaranteed to be a global solution. The presolve algorithm may help the solver in finding a better solution. A local solver might sometimes fail to find a solution and then it is often not clear whether that is caused by the problem being infeasible or

by the solver failing to find a solution for a feasible problem. The presolve algorithm may reveal inconsistent constraints and/or variable bounds and hence identify a problem as infeasible.

Presolve techniques

Consider the following constrained nonlinear optimization problem:

Minimize:

$$f(x)$$

Subject to:

$$\begin{aligned} g(x) &\leq d \\ Ax &\leq b \\ l \leq x &\leq u \end{aligned}$$

The objective function $f(x)$ can either be linear or nonlinear, while $g(x)$ is a nonlinear function. The AIMMS presolve algorithm will (amongst others)

- remove singleton rows by moving the bounds to the variables,
- reduce variable bounds from linear and nonlinear constraints that contain bounded variables,
- delete fixed variables,
- remove one variable of a doubleton, and
- delete redundant constraints.

A detailed description of each of these techniques can be found in [Fo94].

Singleton rows

A singleton row is a linear constraint that contains only one variable. An equality singleton row fixes the variable to the right-hand-side value of the row, and unless this value conflicts with the current bounds of the variable in which case the problem is infeasible, AIMMS can remove both the row and variable from the problem. An inequality singleton row introduces a new bound on the variable which can be redundant, tighter than an existing bound in which case AIMMS will update the bound, or infeasible. The AIMMS presolve algorithm will remove all singleton rows.

Deleting fixed variables

If a variable is fixed then sometimes another row becomes a singleton row, and if that row is an equality row AIMMS can fix the remaining variable and remove it from the problem. By repeating this process AIMMS can solve any triangular system of linear equations that is part of the problem.

Bound reductions using linear constraints

The following analysis applies to a linear "less than or equal to" constraint. A similar analysis applies to other constraint types. Assume we have a linear constraint i that originally has the form

$$\sum_j a_{ij} x_j \leq b_i \tag{17.1}$$

Assuming that all variables in this constraint have finite bounds, we can determine the following lower and upper limits on constraint i

$$\underline{b_i} = \sum_{j \in P_i} a_{ij} l_j + \sum_{j \in N_i} a_{ij} u_j \tag{17.2}$$

and

$$\overline{b_i} = \sum_{j \in P_i} a_{ij} u_j + \sum_{j \in N_i} a_{ij} l_j \tag{17.3}$$

where $P_i = \{j \mid a_{ij} > 0\}$ and $N_i = \{j \mid a_{ij} < 0\}$ define the sets of variables with a positive coefficient and negative coefficient in constraint i respectively.

Bound analysis

By comparing the lower and upper limits of a constraint with the right-hand-side value we obtain one of the following situations:

- $\underline{b_i} > b_i$: constraint (17.1) cannot be satisfied and is infeasible.
- $\underline{b_i} = b_i$: constraint (17.1) can only be satisfied if all variables in the constraint are fixed on their lower bound if they have a positive coefficient, or fixed on their upper bound if they have a negative coefficient. The constraint and all its variables can be removed from the problem.
- $\overline{b_i} \leq b_i$: constraint (17.1) is redundant, i.e. will always be satisfied, and can be removed from the problem.
- $\underline{b_i} < b_i < \overline{b_i}$: constraint (17.1) cannot be eliminated but can often be used to improve the bounds of one or more variables as we will see below.

If $\underline{b_i} < b_i < \overline{b_i}$, then combining (17.1) with (17.2) gives the following bounds on a variable k in constraint i:

$$x_k \leq l_k + (b_i - \underline{b_i})/a_{ik} \qquad \text{if } a_{ik} > 0 \tag{17.4}$$

and

$$x_k \geq u_k + (b_i - \underline{b_i})/a_{ik} \qquad \text{if } a_{ik} < 0 \tag{17.5}$$

If the upper bound given by (17.4) is smaller than the current lower bound of variable k then the problem is infeasible. If it is smaller then the current upper bound of variable k, AIMMS will update the upper bound for variable k. Something similar holds for the lower bound as given by (17.5). Note that bounds (17.4) and (17.5) can only be derived if all bounds l_j and u_j in (17.2) are finite. But also if exactly one of the bounds in (17.2) is an infinite bound, AIMMS can still find an implied bound for the corresponding variable.

Bound reductions using nonlinear constraints

We can rewrite a nonlinear constraint $g_i(x) \leq d_i$ as

$$\sum_j a_{ij} x_i + h_i(y) \leq d_i \tag{17.6}$$

separating the linear variables x in this constraint from the nonlinear variables y. As before, we can find lower and upper limits on the linear part of the constraint, and again we denote them by $\underline{b_i}$ and $\overline{b_i}$ respectively. For this constraint

we can derive the following upper bound on the nonlinear term in (17.6):

$$h_i(y) \leq d_i - \underline{b_i} \tag{17.7}$$

Note that if there are no linear terms in constraint (17.6) then $\underline{b_i} = 0$.

Nonlinear analysis using expression trees

Nonlinear expressions in AIMMS are stored in an expression tree. By going through the expression tree from the top node to the leafs we can sometimes derive bounds on some of the variables in the expression. For example, assume we have the constraint

$$\sqrt{\ln x} \leq 2$$

with x unbounded. It follows that the $\ln x$ sub-expression should be in the range $[0, 4]$ since $\sqrt{y}$ is not defined for $y < 0$, which in turn implies that x should be in the range $(1, e^4]$.

Types of nonlinear analysis

AIMMS can analyze nonlinear expressions for various types of reductions, and uses various types of techniques, such as:

- operator domain analysis: reduce bounds on operator arguments by the implicit domains of operators such as $\sqrt{x}$ or $\ln x$,
- operator range analysis: compute the bounds of a nonlinear expression on the basis of known bounds on the argument(s) and use those bounds for further reductions, and
- for invertible functions, compute bounds on operator arguments on the basis of bounds on a known operator range.

Supported operators

The presolve algorithm can handle nonlinear expressions build up by the operators listed in Table 17.1. If a nonlinear constraint contains an operator that is not in this table then it will be ignored by the presolve algorithm.

$\log_{10} x$, $\ln x$	$\exp x$, e^x
x^a, a^x $(a \neq 0)$	x^y
$\sin x$, $\cos x$, $\tan x$	$\arcsin x$, $\arccos x$, $\arctan x$
$x + y$, $x - y$	$x \cdot y$, x / y

Table 17.1: Operators used by the presolve algorithm

Doubletons

If a problem contains a constraint of the form $x = ay$, $a \neq 0$, then the variables x and y define a doubleton. If the presolve algorithm detects a doubleton then it will replace the variable x by the term ay in every constraint in which x appears, and remove the variable x from the problem. For some problems good initial values are given to the variables. In case the initial value given to x does not match the initial value of y according to the relationship $x = ay$, it is unclear which initial value we should assign to y. Preliminary test results

showed that in such a case it is better not to remove the doubleton, and pass both variables to the solver with their own initial value. This has become the default behavior of our presolve algorithm regarding doubletons.

The presolve algorithm

The AIMMS Presolver iteratively applies all reduction techniques discussed above until no further reductions are available anymore, or an iteration limit has been reached. Various options are available in the **Solvers general - AIMMS presolve** of the option tree to steer the presolve algorithm. For instance a user can choose to only use linear constraints for reducing bounds, or to not remove doubletons.

Mixed integer programming problems

If the optimization problem contains binary variables then the AIMMS Presolver can apply probing which is a technique that looks at the logical implications of fixing a binary variable to 0 or 1. Probing can be used to reduce more variables bounds, reformulate constraints or improve coefficients. In some cases quadratic constraints containing binary variables can be reformulated as linear constraints. Coefficient improvement is a process of improving the coefficients of the binary variables such that the relaxation becomes more tight. A detailed description of probing and coefficient improvement can be found in [Sa94].

Successes may vary

The benefits of using the AIMMS Presolver may vary from model to model. The solution of presolved NLPs may become better or worse compared to the original NLP. Presolving may change infeasible NLPs to feasible problems for a given starting point, or vice versa. Also, presolving may make the model more degenerate and harder to solve. Finaly, for eliminated constraints and variables dual information is lost, and AIMMS makes no effort yet to recover the lost dual information, as this may be very hard in the presence of nonlinear reductions.

17.2 The AIMMS multistart algorithm

The multistart algorithm

A multistart algorithm calls an NLP solver from multiple starting points, keeps track of (all) feasible solutions found by the NLP solver, and reports back the best of these as its final solution.

Why use multistart?

A multistart algorithm can improve the reliability of any NLP solver, by calling it with many starting points. A single call to a NLP solver can fail (return a status of infeasible), but multiple calls from the widely spaced starting points provided by a multistart algorithm have a much better chance of success.

Basic techniques

In a pure multistart algorithm many local searches will converge to the same local minimum. Computational effort can be reduced if the minimizations leading to the same local minimum point can be identified and combined at early stages. An improvement is to use cluster analysis techniques to identify regions of points that will lead to the same local minimum.

Algorithm used by AIMMS

AIMMS uses a multistart algorithm that does not use advanced cluster analysis techniques, but instead tries to identify areas of points that will lead to the same local solution. These areas are updated (and become larger) whenever a starting point is found that leads to a local solution that has already been found before. An more detailed description of a multistart algorithm similar to the one used by AIMMS can be found in [Ka87].

Definitions

The following terminology is used for the multistart algorithm

- Sample points: a set of points that were randomly sampled.
- Cluster point: a point that defines the center of a cluster, i.e., a cluster is a circle/ball with a cluster point as its center.
- Starting point: a point used as an initial solution ("hotstart") for solving the NLP.
- Local solution: a solution found by the NLP solver (by using a starting point). A local solution belongs to exactly one cluster point. A local solution can be infeasible.

Two algorithms

The multistart module implements two algorithms, namely a basic algorithm and a dynamic algorithm in which the number of iterations is changed dynamically. The inputs for both algorithms are:

- a GMP associated with an NLP,
- `NumberOfSamplePoints`, and
- `NumberOfSelectedSamplePoints`.

The basic algorithm

The basic algorithm employs the following steps:

0. Set `IterationCount` equal to 1.
1. Generate `NumberOfSamplePoints` sample points from the uniform distribution. Calculate the penalized objective for all sample points and select the best `NumberOfSelectedSamplePoints` sample points.
2. For all sample points (`NumberOfSelectedSamplePoints` in total) do:
   - For all clusters, calculate the distance between the sample point and the center of the cluster. If the distance is smaller than the radius of the cluster (i.e., the sample point belongs to the cluster) then delete the sample point.
3. For all (remaining) sample points do:
   - Solve the NLP by using the sample point as its starting point to obtain a candidate local solution.

- For all clusters do:
  - Calculate the distance between the candidate local solution and the local solution belonging to the cluster.
  - If the distance equals 0 (which implies that the candidate local solution is the same as the local solution belonging to the cluster) then update the center and radius of the cluster by using the sample point.
  - Else, construct a new cluster by using the mean of the sample point and the candidate local solution as its center with radius equal to half the distance between these two points. Assign the candidate local solution as the local solution belonging to the cluster.
- For all remaining sample points, calculate the distance between the sample point and the center of the updated or the new cluster. If the distance is smaller than the radius of the cluster then delete the sample point.

4. Increment `IterationCount`. If the number of iterations exceeds the `IterationLimit`, then go to step (5). Else go to step (1).
5. Order the local solutions and store the `NumberOfBestSolutions` solutions in the solution repository.

By default, the algorithm uses the initial variable values as the first "sample" point in the first iteration.

The dynamic algorithm: first phase

The dynamic algorithm contains two phases. The first phase is similar to the basic algorithm but with some differences. The dynamic algorithm starts by determining the best sampling box for the creation of the random points (in step 0). For the first sample point, which can be an initial point provided by the user or the first randomly generated point, a method is applied to compute an approximately feasible solution (see [Ch04]) to increase the chance that this first sample point will lead to a feasible solution. Finally, if the dynamic algorithm did not find any feasible solution during the first iterations, and all local solutions found contain large infeasibilities, then a heuristic will be used to update the variable bounds (in step 4).

The dynamic algorithm: second phase

The second phase of the dynamic algorithm is only conducted if no feasible solution was found in the first phase, or if the objective values of the feasible solutions found in the first phase vary. The second phase differs for both situations. If no feasible solution was found in the first phase then the algorithm will continue with steps 1 to 4 until a feasible solution is found or the time limit is hit. In each iteration, the algorithm will now use the method for computing an approximately feasible solution for the first randomly generated point. In the other case, in which the objective values of the feasible solutions found in the first phase vary, the second phase will continue with steps 1 to 4 until enough feasible solutions are found to satisfy a Bayesian estimate for the number of local feasible solutions (or if the time limit is hit).

Using the AIMMS *multistart algorithm*

The AIMMS multistart algorithm is implemented as a system module, with the name Multi Start, that you can add to your project. You can install this module using the **Install System Module** command in the AIMMS **Settings** menu. The algorithm outlined above is implemented in the AIMMS language. Some supporting functions that are computationally difficult, or hard to express in the AIMMS language, have been added to the GMP library in support of the AIMMS multistart algorithm.

Calling the multistart algorithm

The main procedure to start the multistart algorithm is the procedure DoMultiStart. The only mandotory input is a generated mathematical program obtained by calling the GMP::Instance::Generate function of the GMP library discussed in Section 16.2. Therefore the multistart algorithm can be called by using for example:

```
MulStart::DoMultiStart( myGMP );
```

Here MulStart is the prefix of the multistart module. The behavior of the multistart algorithm is influenced by several control parameters, which are discussed in Section 17.3.

Optional arguments

The procedure DoMultiStart contains two optional arguments (with a default value of 0) which can be used to specify the number of sample points and the number of selected sample points (as outlined above). If both arguments are not specified (like in the example of the previous paragraph) or are equal to 0, then the multistart algorithm will use the dynamic algorithm, and otherwise the basic algorithm. For example, if

```
MulStart::DoMultiStart( myGMP, 20, 10 );
```

is used then the basic algorithm will be used with 20 sample points and 10 selected sample points. If the dynamic algorithm is used then the multistart algorithm will automatically select values for the number of sample points and the number of selected sample points. It is possible to use the dynamic algorithm and specify the number of sample points and the number of selected sample points yourself by calling the procedure DoMultiStartDynamic.

Supporting GMP *functions*

The GMP library contains the following functions to support the multistart algorithm:

- GMP::Solution::RandomlyGenerate (used in step (1))
- GMP::Solution::GetPenalizedObjective (used in step (1))
- GMP::Solution::GetDistance (used in steps (2) and (4))
- GMP::Solution::ConstructMean (used in step (4))
- GMP::Solution::UpdatePenaltyWeights (used during initialization)

Optionally it is possible to (approximately) project each sample point to the feasible region by using the procedure GMP::Instance::FindApproximatelyFeasibleSolution.

Modifying the algorithm

Because the multistart algorithm is written in the AIMMS language, you have complete freedom to modify the algorithm in order to tune it for your nonlinear programs.

17.3 Control parameters that influence the multistart algorithm

Control parameters

The multistart module defines several parameters that influence the multistart algorithm. These parameters have a similar functionality as options of a solver, e.g., CPLEX. The most important parameters, with their default setting, are shown in Table 17.2. The parameters that are not self-explanatory are explained in this section; the last column in the table refers to the subsection that discusses the corresponding parameter.

Parameter	Default	Range	Subsection
IterationLimit	5	{1,maxint}	17.3.1
TimeLimit	0	{0,maxint}	17.3.2
TimeLimitSingleSolve	0	{0,maxint}	17.3.2
ThreadLimit	0	{0,maxint}	17.3.3
UseOpportunisticAlgorithm	0	{0,1}	17.3.4
NumberOfBestSolutions	1	{1,1000}	17.3.5
ShrinkFactor	0.95	[0,1]	17.3.6
UsePresolver	1	{0,1}	17.3.7
UseInitialPoint	1	{0,1}	17.3.8
UseConstraintConsensusMethod	0	{-1,1}	17.3.9
MaximalVariableBound	1000	[0,inf)	17.3.10
ShowSolverProgress	0	{0,1}	17.3.11

Table 17.2: Control parameters in the multistart module

17.3.1 Specifying an iteration limit

Parameter IterationLimit

The parameter IterationLimit can be used to set a limit on the number of iterations used by the multistart algorithm. This limit is use in the basic algorithm and in the first phase of the dynamic algorithm.

17.3.2 Specifying a time limit

Parameter TimeLimit

The parameter TimeLimit can be used to set a limit on the total elapsed time (in seconds) used by the multistart algorithm. The default value of 0 has a special meaning; in that case there is no time limit.

It is also possible to set a time limit for every single solve started by the multistart algorithm by using the parameter TimeLimitSingleSolve. Also the default value of 0 of this parameter has a special meaning; in that case there is no time limit.

Parameter TimeLimitSingleSolve

17.3.3 Using multiple threads

The parameter ThreadLimit controls the number of threads that should be used by the multistart algorithm. Each thread will be used to solve one NLP using an asynchronous solver session. At its default setting of 0, the algorithm will automatically use the maximum number of threads, which is limited by the number of cores on the machine and the amount of solver sessions allowed by the AIMMS license.

Parameter ThreadLimit

17.3.4 Deterministic versus opportunistic

By default the multistart algorithm runs in deterministic mode. Deterministic means that multiple runs with the same model using the same parameter settings and the same solver on the same computer will reproduce the same results. The number of NLP problems solved by the multistart algorithm will then also be the same. In contrast, opportunistic implies that the results, and the number of NLP problems solved, might be different. Usually the opportunistic mode provides better performance. The parameter UseOpportunisticAlgorithm can be used to switch to the opportunistic mode. Note that if the multistart algorithm uses only one thread then the algorithm will always be deterministic.

Parameter UseOpportunisticAlgorithm

17.3.5 Getting multiple solutions

By default the multistart algorithm will return one solution, namely the best solution that the algorithm finds. By setting the parameter NumberOfBestSolutions to a value higher than 1, the multistart algorithm will store the best n solutions found in the solution repository (see Section 16.4). Here n denotes the value of this parameter.

Parameter NumberOfBestSolutions

17.3.6 Shrinking the clusters

The clusters created by the multistart algorithm would normally grow as more and more points are assigned to the clusters. As a side effect, a new sample point is then more likely to be directly assigned to a cluster, in which case no NLP is solved for that sample point, thereby increasing the chance that it ends up in the wrong cluster. To overcome this problem, the multistart

Parameter ShrinkFactor

algorithm automatically shrinks all clusters after each iteration by a constant factor which is specified by the parameter ShrinkFactor.

17.3.7 Combining multistart and presolver

Parameter UsePresolver

By default the multistart algorithm starts by applying the AIMMS Presolver to the NLP problem. By preprocessing the problem, the ranges of the variables might become smaller which has a positive effect on the multistart algorithm as then the randomly generated sample points are more likely to be good starting points. The parameter UsePresolver can be used to switch off the preprocessing step.

17.3.8 Using a starting point

Parameter UseInitialPoint

Sometimes the level values, assigned to the variables before solving the NLP problem, provide a good starting point. By default the multistart algorithm will use this initial point as the first sample point but only in the first iteration. This behavior is controlled by the parameter UseInitialPoint.

17.3.9 Improving the sample points

Parameter UseConstraintConsensusMethod

The sample points are randomly generated by using the intervals defined by the lower and upper bounds of the variables. Such a sample point is very likely to be infeasible with respect to the constraints. The constraint consensus method, which is described in [Ch04], tries to find an approximately feasible point for a sample point. Using this method might slow down the multistart algorithm but the chance of generating (almost) feasible sample points increases. The constraint consensus method can be activated by using the parameter UseConstraintConsensusMethod. If this parameter is set to 1 then the constraint consensus method will be used whenever possible, and if it is set to -1 then it will never be used. At its default value of 0, the algorithm automatically decides when to use the constraint consensus method.

17.3.10 Unbounded variables

Parameter MaximalVariableBound

A multistart algorithm requires that all variable bounds are finite. Therefore the multistart algorithm in AIMMS will use a fixed value for all infinite upper and lower variable bounds. This fixed value is specified by the parameter MaximalVariableBound. The value of this parameter might be updated automatically in case the dynamic algorithm is used.

17.3.11 Solver progress

Parameter ShowSolverProgress

By default the progress window will only show general progress information for the multistart algorithm, including the objective value, the number of iterations, the elapsed time, etc. By switching on the parameter `ShowSolverProgress` also progress information by the NLP solver will be displayed. If multiple solver sessions are (asynchronous) executing at the same time then only the progress information of one of them will be shown.

Bibliography

[Ch04] J.W. Chinneck, *The constraint consensus method for finding approximately feasible points in nonlinear programs*, INFORMS Journal on Computing **16** (2004), 255–265.

[Fo94] R. Fourer and D.M. Gay, *Experience with a primal presolve algorithm*, Large Scale Optimization: State of the Art (W.W. Hager, D.W. Hearn, and P.M. Pardalos, eds.), Kluwer Academic Publishers, 1994.

[Ka87] A.H.G. Rinnooy Kan and G.T. Timmer, *Stochastic global optimization methods; part II: Multi level methods*, Mathematical Programming **37** (1987), 57–78.

[Sa94] M.W.P. Savelsbergh, *Preprocessing and probing techniques for mixed integer programming problems*, ORSA Journal on Computing **6** (1994), 445–454.