
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 or order your hard-copy at www.lulu.com/aimms.

Copyright © 1993–2011 by Paragon Decision Technology B.V. All rights reserved.

Paragon Decision Technology B.V.	Paragon Decision Technology Inc.	Paragon Decision Technology Pte.
Schipholweg 1	500 108th Avenue NE	Ltd.
2034 LS Haarlem	Ste. # 1085	80 Raffles Place
The Netherlands	Bellevue, WA 98004	UOB Plaza 1, Level 36-01
Tel.: +31 23 5511512	USA	Singapore 048624
Fax: +31 23 5511517	Tel.: +1 425 458 4024	Tel.: +65 9640 4182
	Fax: +1 425 458 4025	

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

AIMMS is a registered trademark of Paragon Decision Technology B.V. IBM ILOG CPLEX and sc CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Ziena Optimization, Inc. XPRESS-MP is a registered trademark of FICO Fair Isaac Corporation. MOSEK is a registered trademark of Mosek ApS. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation. \TeX , \LaTeX , and $\AMS-\LaTeX$ are trademarks of the American Mathematical Society. LUCIDA is a registered trademark of Bigelow & Holmes Inc. ACROBAT is a registered trademark of Adobe Systems Inc. Other brands and their products are trademarks of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Paragon Decision Technology B.V. The software described in this document is furnished under a license agreement and may only be used and copied in accordance with the terms of the agreement. The documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Paragon Decision Technology B.V.

Paragon Decision Technology B.V. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Paragon Decision Technology B.V., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claims for lost profits, fees or expenses of any nature or kind.

In addition to the foregoing, users should recognize that all complex software systems and their documentation contain errors and omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors shall not be responsible under any circumstances for providing information or corrections to errors and omissions discovered at any time in this book or the software it describes, whether or not they are aware of the errors or omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors do not recommend the use of the software described in this book for applications in which errors or omissions could threaten life, injury or significant loss.

This documentation was typeset by Paragon Decision Technology B.V. using \LaTeX and the LUCIDA font family.

Chapter 17

Advanced Methods for 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.

Problems of nonlinear programs

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

How to counteract?

- 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 a nonlinear 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 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.

This chapter

17.1 The nonlinear presolver

Of all nonlinear solvers in AIMMS only a couple use (limited) preprocessing techniques. Therefore, AIMMS itself has implemented a nonlinear 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

The need for a nonlinear presolver

presolve algorithm may reveal inconsistent constraints and/or variable bounds and hence identify a problem as infeasible.

Consider the following constrained nonlinear optimization problem:

Presolve techniques

Minimize:

$$f(x)$$

Subject to:

$$g(x) \leq d$$

$$Ax \leq b$$

$$l \leq x \leq u$$

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

- 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].

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.

Singleton rows

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.

Deleting fixed variables

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

Bound reductions using linear constraints

$$\sum_j a_{ij}x_j \leq b_i \quad (17.1)$$

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

$$\underline{b}_i = \sum_{j \in P_i} a_{ij} l_j + \sum_{j \in N_i} a_{ij} u_j \quad (17.2)$$

and

$$\overline{b}_i = \sum_{j \in P_i} a_{ij} u_j + \sum_{j \in N_i} a_{ij} l_j \quad (17.3)$$

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.

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

Bound analysis

- $\underline{b}_i > b_i$: 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 :

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

and

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

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 than 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.

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

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

Bound reductions using nonlinear constraints

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):

$$h_i(y) \leq d_i - \underline{b}_i \quad (17.7)$$

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

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 $(1, e^4]$.

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.

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

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

Nonlinear analysis using expression trees

Types of nonlinear analysis

Supported operators

Doubletons

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 AIMMS nonlinear 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-Nonlinear 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.

The presolve algorithm

The benefits of using the nonlinear 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. Finally, 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.

Successes may vary

17.2 The AIMMS 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. The AIMMS multistart algorithm also has the option to store the n best solutions of an NLP problem in the solution repository (see Section 21.4).

The multistart algorithm

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.

Why use multistart?

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.

Basic techniques

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

Algorithm used by AIMMS

found before. An more detailed description of a multistart algorithm similar to the one used by AIMMS can be found in [Ka87].

The following terminology is used for the multistart algorithm

Definitions

- 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 to solve 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.

The inputs for the algorithm are:

Basic algorithm

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

The algorithm employs the following steps:

1. Set `NumberOfIterations` equal to 1.
2. Generate `NumberOfSamplePoints` sample points from uniform distribution. Calculate the penalized objective for all sample points and select the best `NumberOfSelectedSamplePoints` sample points.
3. Take one of the `NumberOfSelectedSamplePoints` sample points. If all sample points are processed then go to step (7).
4. 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 go to step (3).
5. Solve NLP by using the sample point as its starting point to obtain a candidate local solution. For all clusters, calculate the distance between the candidate local solution and the local solution belonging to the cluster. If the distance is 0 then the candidate local solution is the same as the local solution belonging to the cluster; update the center and radius of the cluster by using the sample point and go to step (3).
6. Construct 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. Go to step (3).
7. Increment `NumberOfIterations`. If the number of iterations exceeds the `IterationLimit`, then go to step (8). Else go to step (2).
8. Order the local solutions and store the `NumberOfBestSolutions` solutions in the solution repository.

The algorithm starts in fact by solving the NLP once by using the starting point (i.e., doing a normal solve). The starting point and the solution obtained are used to create the first cluster, as in step (6).

The AIMMS multistart algorithm is implemented as a system module that you can add to your model. You can add the multistart module to your model, through the **Settings-Install System Module...** menu in the model editor. 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.

Using the AIMMS multistart algorithm

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

Supporting GMP functions

- `GMP::Solution::RandomlyGenerate` (used in step (2))
- `GMP::Solution::GetPenalizedObjective` (used in step (2))
- `GMP::Solution::GetDistance` (used in steps (4) and (5))
- `GMP::Solution::ConstructMean` (used in steps (5) and (6))
- `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`.

The main procedure to start the multistart algorithm is the procedure `DoMultiStart`. Its inputs are a generated mathematical program obtained by calling the `GMP::Instance::Generate` function of the GMP library discussed in Section 21.2, the number of sample points, and the number of selected sample points (as outlined above).

Calling the multistart 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. Also, the basic algorithm solves all NLPs sequentially. If your AIMMS license permits parallel solver sessions, you can also speed up the algorithm by solving multiple NLPs in parallel using the GMP function `GMP::SolverSession::AsynchronousExecute`.

Modifying the algorithm

Bibliography

- [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.