
AIMMS Function Reference - AIMMS Outer Approximation Functions

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

Outer Approximation Functions

The AIMMS Outer Approximation functions allow you to solve MINLP problems through a sequence of MIP and NLP solves. The following Outer Approximation functions are available.

AIMMS supports the following Outer Approximation functions for solving and managing the master MIP problem:

Master MIP functions

- MasterMIPAddLinearizations
- MasterMIPDeleteIntegerEliminationCut
- MasterMIPDeleteLinearizations
- MasterMIPEliminateIntegerSolution
- MasterMIPGetCPUTime
- MasterMIPGetIterationCount
- MasterMIPGetNumberOfColumns
- MasterMIPGetNumberOfNonZeros
- MasterMIPGetNumberOfRows
- MasterMIPGetObjectiveValue
- MasterMIPGetProgramStatus
- MasterMIPGetSolverStatus
- MasterMIPGetSumOfPenalties
- MasterMIPIsFeasible
- MasterMIPLinearizationCommand
- MasterMIPSetCallback
- MasterMIPsolve

AIMMS supports the following Outer Approximation functions for managing the global MINLP problem:

MINLP functions

- MINLPGetIncumbentObjectiveValue
- MINLPGetOptimizationDirection
- MINLPIncumbentIsFeasible
- MINLPIncumbentSolutionHasBeenFound
- MINLPSetIncumbentSolution
- MINLPSetIterationCount
- MINLPSetProgramStatus
- MINLPsolutionDelete
- MINLPsolutionRetrieve

- `MINLPSolutionSave`

AIMMS supports the following Outer Approximation functions for managing *NLP functions* and solving the NLP problem:

- `NLPGetCPUtime`
- `NLPGetIterationCount`
- `NLPGetNumberOfColumns`
- `NLPGetNumberOfNonZeros`
- `NLPGetNumberOfRows`
- `NLPGetObjectiveValue`
- `NLPGetProgramStatus`
- `NLPGetSolverStatus`
- `NLPisFeasible`
- `NLPLinearizationPointHasBeenFound`
- `NLPSolutionIsInteger`
- `NLPSolve`

MasterMIPAddLinearizations

The procedure `MasterMIPAddLinearizations` adds a linearization for a subset of `AllNonlinearConstraints`. The linearizations are created by using the solution present at that time inside the AIMMS Outer Approximation solver interface. Normally the solution that is returned by the NLP solver is used. When permitted, variables are introduced to allow for deviations from each linearized constraint. These deviation variables are penalized in the objective function using the penalty multipliers times the corresponding shadow prices (Lagrange multipliers). The procedure returns the updated linearization counter in the output argument `n`.

```
MasterMIPAddLinearizations(
    IncludedConstraints, ! (input) subset of the set AllNonlinearConstraints
    DeviationsPermitted, ! (input) 0-1 parameter over AllNonlinearConstraints
    PenaltyMultiplier, ! (input) parameter over AllNonlinearConstraints
    n                   ! (output) integer scalar parameter
)
```

Arguments:

IncludedConstraints

Set of nonlinear constraints for which linearizations have to be added.

DeviationsPermitted

Parameter that indicates whether or not variables should be introduced to allow for deviations from each linearized constraint. If so, the corresponding entry in this parameter should be 1, otherwise 0.

PenaltyMultiplier

The deviation variables (if any) are penalized in the objective function by using the values in this parameter times the corresponding shadow prices (Lagrange multipliers).

n

The updated linearization counter.

Return value:

`MasterMIPAddLinearizations` has no return value.

MasterMIPDeleteIntegerEliminationCut

The procedure `MasterMIPDeleteIntegerEliminationCut` deletes a set of integer solution elimination cuts and variables that was previously added by the `MasterMIPEliminateIntegerSolution` procedure.

```
MasterMIPDeleteIntegerEliminationCut(  
    n      ! (input) integer scalar value  
)
```

Arguments:

n

The cut counter for the set of cuts and variables that has to be deleted. It was returned by `MasterMIPEliminateIntegerSolution` when these cuts and variables were added.

Return value:

`MasterMIPDeleteIntegerEliminationCut` has no return value.

MasterMIPDeleteLinearizations

The procedure `MasterMIPDeleteLinearizations` deletes a set of linearizations that was previously added by the `MasterMIPAddLinearizations` procedure for a certain solution.

```
MasterMIPDeleteLinearizations(  
    n      ! (input) integer scalar value  
)
```

Arguments:

n

The linearization counter for the set of linearizations that has to be deleted. It was returned by `MasterMIPAddLinearizations` when these linearizations were added.

Return value:

`MasterMIPDeleteLinearizations` has no return value.

MasterMIPEliminateIntegerSolution

The procedure `MasterMIPEliminateIntegerSolution` adds a set of cuts and variables to the master MIP model instance which eliminates the current integer solution inside the AIMMS Outer Approximation solver interface.

```
MasterMIPEliminateIntegerSolution(  
    n      ! (output) integer scalar parameter  
)
```

Arguments:

n

The updated cut counter.

Return value:

`MasterMIPEliminateIntegerSolution` has no return value.

Remarks:

To eliminate the current integer solution, 3 variables (2 continuous, 1 binary) and 3 constraints are added for each integer variable whose level value is between its bounds. Also one main cut constraint is added. In case all integer variables are binary, only this main cut constraint is added.

MasterMIPGetCPUtime

The function `MasterMIPGetCPUtime` returns the CPU time needed to solve the master MIP problem.

`MasterMIPGetCPUtime`

Arguments:

None

Return value:

The function `MasterMIPGetCPUtime` returns the double value of the CPU time (in seconds) needed to solve the last master MIP problem.

MasterMIPGetIterationCount

The function `MasterMIPGetIterationCount` returns the iteration count associated with the last master MIP problem solved.

`MasterMIPGetIterationCount`

Arguments:

None

Return value:

The function `MasterMIPGetIterationCount` returns the iteration count associated with the last master MIP problem solved.

MasterMIPGetNumberOfColumns

The function `MasterMIPGetNumberOfColumns` returns the number of columns in the last master MIP problem solved.

`MasterMIPGetNumberOfColumns`

Arguments:

None

Return value:

The function `MasterMIPGetNumberOfColumns` returns the number of columns in the last master MIP problem solved.

MasterMIPGetNumberOfNonZeros

The function `MasterMIPGetNumberOfNonZeros` returns the number of nonzeros in the last master MIP problem solved.

`MasterMIPGetNumberOfNonZeros`

Arguments:

None

Return value:

The function `MasterMIPGetNumberOfNonZeros` returns the number of nonzeros in the last master MIP problem solved.

MasterMIPGetNumberOfRows

The function `MasterMIPGetNumberOfRows` returns the number of rows in the last master MIP problem solved.

`MasterMIPGetNumberOfRows`

Arguments:

None

Return value:

The function `MasterMIPGetNumberOfRows` returns the number of rows in the last master MIP problem solved.

MasterMIPGetObjectiveValue

The function `MasterMIPGetObjectiveValue` returns the objective value of the last solved master MIP.

`MasterMIPGetObjectiveValue`

Arguments:

None

Return value:

The function `MasterMIPGetObjectiveValue` returns the objective value of the last solved master MIP.

MasterMIPGetProgramStatus

The function `MasterMIPGetProgramStatus` returns the program (or model) status associated with the last master MIP problem solved.

`MasterMIPGetProgramStatus`

Arguments:

None

Return value:

The function `MasterMIPGetProgramStatus` returns the program (or model) status associated with the last master MIP problem solved. The return value will be an element in the set `AllSolutionStates`.

MasterMIPGetSolverStatus

The function `MasterMIPGetSolverStatus` returns the solver status associated with the last master MIP problem solved.

`MasterMIPGetSolverStatus`

Arguments:

None

Return value:

The function `MasterMIPGetSolverStatus` returns the solver status associated with the last master MIP problem solved. The return value will be an element in the set `AllSolutionStates`.

MasterMIPGetSumOfPenalties

The function `MasterMIPGetSumOfPenalties` returns the sum of the penalties in the solution of the last solved master MIP.

`MasterMIPGetSumOfPenalties`

Arguments:

None

Return value:

The function `MasterMIPGetSumOfPenalties` returns the sum of the penalties in the solution of the last solved master MIP.

MasterMIPsFeasible

The function `MasterMIPsFeasible` indicates whether the solution found for the last solved master MIP is feasible or not.

`MasterMIPsFeasible`

Arguments:

None

Return value:

The function `MasterMIPsFeasible` returns 1 if the solution of the last master MIP is feasible, or 0 otherwise.

MasterMIPLinearizationCommand

The procedure `MasterMIPLinearizationCommand` allows you to retrieve or modify certain aspects of the linearization of a constraint added for linearization counter n at the individual level. The argument `Command` specifies which data (e.g. `GetDeviation`) should be retrieved or modified. The retrieved or modified value is passed through the `CommandData` argument.

```
MasterMIPLinearizationCommand(
  n,                ! (input) integer scalar value
  ModelConstraint, ! (input) scalar value
  Command,          ! (input) element parameter into
                  ! MasterMIPLinearizationCommands
  CommandData      ! (inout) scalar value (in) or parameter (out)
)
```

Arguments:

n

The linearization counter as returned by `MasterMIPAddLinearizations` when adding this linearization.

ModelConstraint

Scalar reference to a constraint for which certain aspects of the linearization have to be retrieved or modified.

Command

Element parameter into `MasterMIPLinearizationCommands` that specifies which data should be retrieved or modified. Possible values are:

Command	Description
<code>GetDeviation</code>	Get the value of the deviation variable.
<code>RemoveDeviation</code>	Delete the deviation variable.
<code>GetWeight</code>	Get the objective coefficient of the deviation variable.
<code>SetWeight</code>	Set the objective coefficient of the deviation variable.
<code>GetDeviationBound</code>	Get the upper bound of the deviation variable.
<code>SetDeviationBound</code>	Get the upper bound of the deviation variable.
<code>GetLagrangeMultiplier</code>	Get value of the shadow price (Lagrange multiplier) of constraint for last solved NLP.

CommandData

The retrieved or modified value.

Return value:

`MasterMIPLinearizationCommand` has no return value.

Remarks:

- Normally, the weight obtained with 'GetWeight' equals the value of the penalty multiplier, as passed to `MasterMIPAddLinearizations`, times the shadow price (Lagrange multiplier) of the constraint. With 'SetWeight' this weight can be changed.
- Note that 'SetWeight' can be used to create a deviation variable (slack) if the linearization does not have one. To do so the value filled in for `CommandData` should be unequal to 0.
- The lower bound of a deviation variable always equals 0.

MasterMIPSetCallback

The procedure `MasterMIPSetCallback` allows the user to set a callback procedure that will be called during the solve of the master MIP. It will be called either for every new incumbent value found by the MIP solver or after a certain number of iterations. This is determined by the argument `Iterations`.

```
MasterMIPSetCallback(  
    ProcedureName,      ! (input) scalar string expression  
    Iterations          ! (input) integer scalar value  
)
```

Arguments:

ProcedureName

The name of the AIMMS procedure that will be used as callback procedure.

Iterations

If $\text{Iterations} \geq 1$ then the callback procedure will be called after this number of iterations; else it will be called for every new incumbent value found by the MIP solver.

Return value:

`MasterMIPSetCallback()` has no return value.

MasterMIPsolve

The procedure `MasterMIPsolve` calls the MIP solver to solve the master MIP problem. Any modifications that have been made since the last call to `MasterMIPsolve` will be added to the master MIP prior to solving. Examples of such modifications are additions of linearizations and cuts that eliminate integer solutions.

`MasterMIPsolve`

Arguments:

None

Return value:

`MasterMIPsolve()` has no return value.

MINLPGetIncumbentObjectiveValue

The function `MINLPGetIncumbentObjectiveValue` returns the objective value associated with the incumbent solution.

`MINLPGetIncumbentObjectiveValue`

Arguments:

None

Return value:

The function `MINLPGetIncumbentObjectiveValue` returns the objective value associated with the incumbent solution.

MINLPGetOptimizationDirection

The function `MINLPGetOptimizationDirection` returns the optimization direction: 1 for maximization and -1 for minimization.

`MINLPGetOptimizationDirection`

Arguments:

None

Return value:

The function `MINLPGetOptimizationDirection` returns 1 for maximization and -1 for minimization.

MINLPIncumbentIsFeasible

The function `MINLPIncumbentIsFeasible` indicates whether the current incumbent solution is feasible or not for the MINLP problem.

`MINLPIncumbentIsFeasible`

Arguments:

None

Return value:

The function `MINLPIncumbentIsFeasible` returns 1 if the current incumbent solution is feasible for the MINLP problem, or 0 otherwise.

MINLPIncumbentSolutionHasBeenFound

The function `MINLPIncumbentSolutionHasBeenFound` indicates whether an incumbent has already been specified.

`MINLPIncumbentSolutionHasBeenFound`

Arguments:

None

Return value:

The function `MINLPIncumbentSolutionHasBeenFound` returns 1 if an incumbent has already been specified, or 0 otherwise.

MINLPSetIncumbentSolution

The procedure `MINLPSetIncumbentSolution` marks the current values of the decision variables as an incumbent solution for the MINLP problem.

`MINLPSetIncumbentSolution`

Arguments:

None

Return value:

`MINLPSetIncumbentSolution()` has no return value.

MINLPSetIterationCount

The procedure `MINLPSetIterationCount` sets the iteration count for the MINLP problem.

```
MINLPSetIterationCount(  
    IterationCount    ! (input) integer scalar value  
)
```

Arguments:

IterationCount

The iteration number that should be set for the MINLP problem.

Return value:

`MINLPSetIterationCount()` has no return value.

MINLPSetProgramStatus

The procedure `MINLPSetProgramStatus` sets the program status for the MINLP problem.

```
MINLPSetProgramStatus(  
    ProgramStatus    ! (input) element parameter into AllSolutionStates  
)
```

Arguments:

ProgramStatus
Element parameter into `AllSolutionStates` that sets the program status for the MINLP problem.

Return value:

`MINLPSetProgramStatus()` has no return value.

MINLPSolutionDelete

The procedure `MINLPSolutionDelete` deletes the solution inside the AIMMS Outer Approximation solver interface that was previously saved by a call to `MINLPSolutionSave` with solution number n .

```
MINLPSolutionDelete(  
    n      ! (input) integer scalar value  
)
```

Arguments:

n

The solution number corresponding to the solution that has to be deleted. The solution number was passed to `MINLPSolutionSave` before to label the solution.

Return value:

`MINLPSolutionDelete` has no return value.

MINLPSolutionRetrieve

The procedure `MINLPSolutionRetrieve` retrieves the solution previously saved by a call to `MINLPSolutionSave` with solution number `n`, and stores it as the current solution inside the AIMMS Outer Approximation solver interface.

```
MINLPSolutionRetrieve(  
    n      ! (input) integer scalar value  
)
```

Arguments:

n

The solution number corresponding to the solution that has to be retrieved. The solution number was passed to `MINLPSolutionSave` before to label the solution.

Return value:

`MINLPSolutionRetrieve` has no return value.

MINLPSolutionSave

The procedure `MINLPSolutionSave` saves the current solution that is present inside the AIMMS Outer Approximation solver interface, and stores it as solution number n for later retrieval.

```
MINLPSolutionSave(  
    n      ! (input) integer scalar value  
)
```

Arguments:

n

The solution number used to label the saved solution.

Return value:

`MINLPSolutionSave` has no return value.

Remarks:

If a solution was saved before with the same value for n then that solution will be replaced by this new solution.

NLPGetCPUtime

The function `NLPGetCPUtime` returns the CPU time needed to solve the last NLP subproblem.

`NLPGetCPUtime`

Arguments:

None

Return value:

The function `NLPGetCPUtime` returns the double value of the CPU time (in seconds) needed to solve the last NLP subproblem.

NLPGetIterationCount

The function `NLPGetIterationCount` returns the iteration count associated with the last NLP subproblem solved.

`NLPGetIterationCount`

Arguments:

None

Return value:

The function `NLPGetIterationCount` returns the iteration count associated with the last NLP subproblem solved.

NLPGetNumberOfColumns

The function `NLPGetNumberOfColumns` returns the number of columns in the last NLP subproblem solved.

`NLPGetNumberOfColumns`

Arguments:

None

Return value:

The function `NLPGetNumberOfColumns` returns the number of columns in the last NLP subproblem solved.

NLPGetNumberOfNonZeros

The function `NLPGetNumberOfNonZeros` returns the number of nonzeros in the last NLP subproblem solved.

`NLPGetNumberOfNonZeros`

Arguments:

None

Return value:

The function `NLPGetNumberOfNonZeros` returns the number of nonzeros in the last NLP subproblem solved.

NLPGetNumberOfRows

The function `NLPGetNumberOfRows` returns the number of rows in the last NLP subproblem solved.

`NLPGetNumberOfRows`

Arguments:

None

Return value:

The function `NLPGetNumberOfRows` returns the number of rows in the last NLP subproblem solved.

NLPGetObjectiveValue

The function `NLPGetObjectiveValue` returns the objective value of the last solved NLP.

`NLPGetObjectiveValue`

Arguments:

None

Return value:

The function `NLPGetObjectiveValue` returns the objective value of the last solved NLP.

NLPGetProgramStatus

The function `NLPGetProgramStatus` returns the program (or model) status associated with the last NLP subproblem solved.

`NLPGetProgramStatus`

Arguments:

None

Return value:

The function `NLPGetProgramStatus` returns the program (or model) status associated with the last NLP subproblem solved. The return value will be an element in the set `AllSolutionStates`.

NLPGetSolverStatus

The function `NLPGetSolverStatus` returns the solver status associated with the last NLP subproblem solved.

`NLPGetSolverStatus`

Arguments:

None

Return value:

The function `NLPGetSolverStatus` returns the solver status associated with the last NLP subproblem solved. The return value will be an element in the set `AllSolutionStates`.

NLPisFeasible

The function `NLPisFeasible` indicates whether the solution found for the last solved NLP is feasible or not.

`NLPisFeasible`

Arguments:

None

Return value:

The function `NLPisFeasible` returns 1 if the solution of the last NLP is feasible, or 0 otherwise.

NLPLinearizationPointHasBeenFound

The function `NLPLinearizationPointHasBeenFound` indicates whether the NLP solver has found a point that can be used to linearize the nonlinear constraints. If the NLP problem is infeasible then usually the NLP solver provides a point that solves the so-called feasibility problem (i.e., a point that minimizes the sum of the infeasibilities).

`NLPLinearizationPointHasBeenFound`

Arguments:

None

Return value:

The function `NLPLinearizationPointHasBeenFound` returns 1 if the NLP solver has found a point that can be used to linearize the nonlinear constraints. It returns 0 otherwise.

Remarks:

This function always returns 1 if the NLP has found a feasible solution.

NLPSolutionIsInteger

The function `NLPSolutionIsInteger` indicates whether the solution found for the last NLP is integer and feasible, or not.

`NLPSolutionIsInteger`

Arguments:

None

Return value:

The function `NLPSolutionIsInteger` returns 1 if the solution of the last NLP is integer feasible, or 0 otherwise.

NLPSolve

The procedure `NLPSolve` calls the NLP solver to solve the NLP subproblem in which the (symbolic) integer variables in the set `FrozenVariables` remain frozen during the solve, and all other integer variables are considered to be continuous between their bounds.

```
NLPSolve(  
    FrozenVariables    ! (input) subset of the set AllIntegerVariables  
)
```

Arguments:

FrozenVariables

The set of (symbolic) integer variables that remain frozen during the solve of the NLP.

Return value:

`NLPSolve()` has no return value.