AIMMS Language Reference - Robust Optimization

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.

Chapter 20

Robust Optimization

Robust optimization is a rather new modeling methodology for decision making under uncertainty. Robust optimization is designed to meet some major challenges associated with uncertainty-affected optimization problems:

Introduction

- to operate under lack of full information on the nature of uncertainty,
- to model the problem in a form that can be solved efficiently, and
- to provide guarantees about the performance of the solution.

Robustness of decisions is defined in terms of the best performance in the worst case possible state-of-the-world (min-max optimization). A more in-depth discussion of robust optimization can be found, for instance, in [BT09].

In this chapter, you will find a description of the facilities built into Aimms for creating and solving robust optimization models. From any existing deterministic linear program (LP) or mixed-integer program (MIP), Aimms is able to automatically create a robust optimization model as well, *without the need for you to reformulate any of the constraint definitions*. The only steps necessary to create a robust optimization model are

Robust optimization in Aimms

- to indicate which parameters in your deterministic model are to become uncertain in a declarative manner,
- to indicate which variables in your deterministic model are to become adjustable to the uncertain parameters (if any), and
- to specify possible realizations of the uncertain parameters.

Being able to generate both a deterministic and robust optimization model from an identical symbolic formulation allows for any changes you make in the deterministic formulation to automatically propagate to the robust optimization model. This significantly reduces the effort involved with maintaining a robust optimization model associated with a given deterministic model.

Single formulation

To be able to run an robust optimization model, you need to make sure you have the *Robust Optimization Add-On* licensed. Without the RO Add-On, you can still define your robust optimization models, but will be unable to solve them (an execution error will occur).

Robust Optimization Add-On required

The Robust Optimization Add-On in AIMMS has been developed in close co-operation with Professor Aharon Ben-Tal and Boris Bachelis of the Technion, Israel Institute of Technology. We would like to express our gratitude for our partnership in developing the Robust Optimization Add-On in AIMMS and for their continuous support to get the details right, which allowed us to make Robust Optimization a natural and intuitive extension to our existing functionality.

Acknowledge-ments

Section 20.1 discusses a number of basic concepts in robust optimization. These provide a common understanding necessary for the introduction of the robust optimization features of AIMMS discussed in the sections to follow. Section 20.2 describes the facilities available in AIMMS for specifying uncertain parameters, while Section 20.3 discusses chance constraints as another means to introduce uncertainty into your robust optimization model. Section 20.4 discusses the facilities available to declare variables to be adjustable to uncertain parameters. Section 20.5, finally, describes the steps how to actually solve a robust optimization model.

This chapter

20.1 Basic concepts

In this section you will find a number of basic concepts that are commonly used in robust optimization. They will help you to unambiguously understand the robust optimization facilities in AIMMS.

Basic concepts

In robust optimization the model with uncertain data is translated into the so-called *robust counterpart*. Consider the following linear programming problem:

Robust counterpart

$$\max\{c^T x : A^T x \leq b\} \qquad \text{(P)}$$

in which $c \in \mathbb{R}^m$, $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$. Suppose that the actual technology matrix A is in fact uncertain and it is only known to belong to a bounded uncertainty set $U_A \subset \mathbb{R}^{m \times n}$. Similarly assume that right hand side b belongs to an uncertainty set $U_b \subset \mathbb{R}^n$, and the objective coefficients c to an uncertainty set $U_c \subset \mathbb{R}^m$. The robust counterpart (RC) for the nominal problem (P) is then defined as follows:

$$\max\{c^T x : A^T x \leq b, \ \forall A \in U_A, c \in U_c, b \in U_b\}. \qquad \text{(RC)}$$

The sets U_A, U_c and U_b specify all possible realizations of the uncertain data and are collectively called the *uncertainty set*. The main questions associated with the uncertainty set are:

Uncertainty set

- When and how can the robust counterpart of an uncertain problem be reformulated as a computationally tractable optimization problem?

■ How to specify a reasonable uncertainty set, i.e., meaningful for a particular application and yielding a tractable robust counterpart?

It can be shown that when the uncertainty set is a multi-dimensional interval or described by linear constraints, then the robust counterpart can be reformulated as a linear problem. Furthermore, when the uncertainty set is an ellipsoid, then the robust counterpart is still tractable, i.e., it can be reformulated as a second-order cone program (SOCP), for which efficient (polynomial time) solution methods exist. The reformulation of the robust counterpart is an automated process performed by AIMMS during the generation of your mathematical program.

If the uncertainty set is a multi-dimensional interval or described by linear constraints, then the robust counterpart of a mixed-integer robust optimization problem can also be reformulated as a mixed-integer optimization problem. If the uncertainty set is described by ellipsoidal constraints then the robust counterpart becomes a mixed-integer second-order cone program. This class of problems is more difficult to solve than mixed-integer optimization problems.

Integer programming

A special situation to consider is when the uncertainty set consists of a finite number of points representing a collection of scenarios. This discrete case resembles the situation in multi-stage stochastic programming with discrete data realizations. More precisely, in this case hard constraints are imposed for every scenario s, while the objective is to optimize a worst-case performance measure with respect to the set of scenarios. This performance measure can be, for example, the objective value of the original (deterministic) model associated with an uncertain scenario. Another possibility is to define this performance measure as a deviation of the objective for a decision with respect to the absolutely optimal objective for each scenario. In the latter case, the optimal robust solution will be the one with the minimum maximum deviation across scenarios.

Scenarios

Another manner to account for uncertainty into your model is by specifying so-called *chance constraints*. In order to introduce chance constraints, you have to declare some of the parameters in your model to take random values from a distribution with given characteristics. Subsequently, you can specify that some of the constraints in your model be satisfied with a given probability with respect to the specified data distributions. For example, if a chance constraint has a probability of 95%, this means that the constraint should be satisified for (at least) 95% of the realizations drawn from specified distributions of the random parameters contained in it. Compared to using uncertain parameters, specifying random parameters with the same range and formulating the existing constraints as chance constraints may lead to solutions that put less emphasis on worst-case scenarios that only occur occasionally.

Chance constraints

All decision variables in problem (P) represent "here and now" decisions; they get specific numerical values as a result of solving the problem before the actual data "reveals itself" and as such are independent of the actual values of the data. There are situations where this is too restrictive, since "in reality" some of the decision variables can adjust themselves, to some extent, to the true values of the uncertain data.

Multistage optimization

For that reason, it is possible to specify both *non-adjustable* and *adjustable* variables in AIMMS, similar to first-stage and second-stage (or multi-stage) decisions in stochastic programming, where the solution of a variable in stage n depends on the specific solution of variables in stage $n - 1$ in a scenario-dependent manner. Please note that, while non-adjustable variables can be integer, adjustable variables *must* be continuous.

Adjustable variables

The implementation of robust optimization in AIMMS closely follows the concepts described in this section. The basic procedure to create and solve a robust optimization model in AIMMS is as follows:

Basic procedure for solving robust optimization models

- indicate in your model which parameters are to become uncertain or random,
- for every constraint in your model that you want to become a chance constraint, specify the probability with which it must hold,
- for every adjustable variable in your model specify on which uncertain parameters it depends, and
- specify possible realizations of the uncertain parameters in terms of predefined regions or using specialized uncertainty constraints.

Each of these steps is explained in more detail in the sections to follow. Note that changing parameters, variables and constraints in your model into uncertain or random parameters, adjustable variables and chance constraints does in no way influence the possibility to solve the underlying deterministic model in its original form. Thus, the robust optimization facilities in AIMMS always form a true extension of the functionality of the existing AIMMS application. It is even possible to extend an existing deterministic model to both a stochastic model and a robust optimization model, all of which can be solved independently.

20.2 Uncertain parameters and uncertainty constraints

Uncertain parameters are modeled in AIMMS as numeric Parameters for which the Uncertain property has been set (see also Section 4.1). When a parameter has been declared Uncertain AIMMS will create two new attributes Region and Uncertainty.

Uncertain parameters

The `Region` attribute of an uncertain parameter offers an easy way to define the uncertainty set without the need to introduce additional uncertain parameters. AIMMS supports a number of predefined regions which you can enter here:

The Region attribute

- `Box`(l, u),
- `Ellipsoid`(c, r), and
- `ConvexHull`$(s, v(s))$.

If we want to specify that parameter `A` is uncertain and constrained as follows:

Box example

$$l(i, j) \le A(i, j) \le u(i, j)$$

then it suffices to specify the uncertainty set of `A` using its `Region` attribute as follows

```
Parameter A {
    IndexDomain  : (i,j);
    Property     : Uncertain;
    Region       : Box( l(i,j), u(i,j) );
}
```

where `l(i,j)` and `u(i,j)` are ordinary parameters in your model.

It is also possible to specify the region using an `Ellipsoid` region

Ellipsoid example

```
Parameter A {
    IndexDomain  : (i,j);
    Property     : Uncertain;
    Region       : Ellipsoid( A.level(i,j), r(i,j) );
}
```

which leads to an uncertainty set for `A` defined as an ellipsoid around the nominal value of `A` as follows:

$$\sum_{i,j} \left(\frac{A(i, j) - A.level(i, j)}{r(i, j)} \right)^2 \le 1.$$

The region can also be defined as a `ConvexHull` region

ConvexHull example

```
Parameter A {
    IndexDomain  : (i,j);
    Property     : Uncertain;
    Region       : ConvexHull( s, A_s(s,i,j) );
}
```

which says that the uncertain parameter `A` belongs to an uncertainty set that is described by the convex hull of the values of a collection of values `A_s` for a given set of scenarios, i.e.,

$$A(i, j) = \sum_s \lambda_s A_s(s, i, j)$$

$$1 = \sum_s \lambda_s, \qquad \lambda_s \ge 0.$$

If there are two parameters A and B that both depend on scenario-dependent   *Dependencies*
data, then those scenarios can either be dependent or independent. To differ-
entiate between these two possibilities, AIMMS uses the name of the binding
index used in the ConvexHull operator. If the names of the binding indices are
identical, then AIMMS assumes that the scenarios are dependent. If the index
names are different, *even if they refer to the same scenario set*, AIMMS assumes
the scenarios to be independent.

Consider the following two declarations of uncertain parameters   *Dependent*
scenarios
example

```
Parameter A {
    IndexDomain  :  (i,j);
    Property     :  Uncertain;
    Region       :  ConvexHull( s, A_s(s,i,j) );
}
Parameter B {
    IndexDomain  :  (i,j);
    Property     :  Uncertain;
    Region       :  ConvexHull( s, B_s(s,i,j) );
}
```

Based on these declarations AIMMS will generate a single convex hull as follows

$$\begin{bmatrix} A(i,j) \\ B(i,j) \end{bmatrix} = \sum_s \lambda_s \begin{bmatrix} A_s(s,i,j) \\ B_s(s,i,j) \end{bmatrix}$$

$$\sum_s \lambda_s = 1, \qquad \lambda_s \geq 0.$$

If A and B consist of a single value each, and there are two scenarios for s, then
the combined convex hull for A and B is depicted in Figure 20.1.
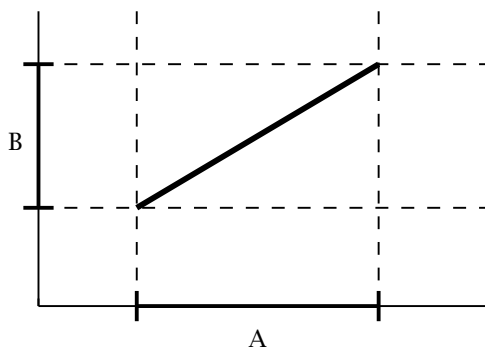
Figure 20.1: Combined convex hull for dependent scenarios

If, on the other hand, both declarations are given as

```
Parameter A {
    IndexDomain  :  (i,j);
    Property     :  Uncertain;
    Region       :  ConvexHull( s, A_s(s,i,j) );
}
Parameter B {
    IndexDomain  :  (i,j);
    Property     :  Uncertain;
    Region       :  ConvexHull( t, B_t(t,i,j) );
}
```

then AIMMS will generate two separate convex hulls as follows

$$\begin{bmatrix} A(i,j) \\ B(i,j) \end{bmatrix} = \begin{bmatrix} \sum_s \lambda_s A_s(s,i,j) \\ \sum_t \mu_t B_t(t,i,j) \end{bmatrix}$$
$$\sum_s \lambda_s = \sum_t \mu_t = 1, \qquad \lambda_s \geq 0, \mu_t \geq 0.$$

If A and B consist of a single value each, and there are two scenarios for s and t each, then the combined convex hull for A and B is depicted in Figure 20.2.
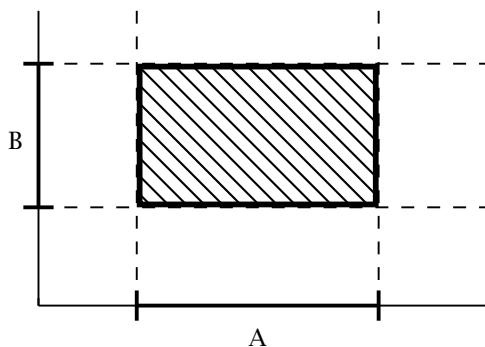
Figure 20.2: Combined convex hull for independent scenarios

The ConvexHull operator AIMMS can be used to express that an uncertain parameter is defined as the convex combination of a certain parameter on some set of scenarios. The ConvexHullEx operator is an extension for which the user explicitly has to define the "lambda" parameter as an uncertain parameter. For example:

```
Parameter A {
    IndexDomain  : (i,j);
    Property     : Uncertain;
    Region       : ConvexHullEx( s, A_s(s,i,j), L(s,i) );
}
```

which says that the uncertain parameter A belongs to an uncertainty set that is described by the convex hull of the values of a collection of values A_s for a

given set of scenarios using the uncertain parameter L, i.e.,

$$A(i,j) = \sum_s L_s(i)A_s(s,i,j)$$

$$1 = \sum_s L_s(i), \qquad L_s(i) \geq 0.$$

The ConvexHullEx operator offers more flexibility as demonstrated by the above example in which the *lambda* parameter L depends on the indices s and i while the implicitly generated *lambda* parameter in case of the ConvexHull operator only depends on the index s. Moreover, the *lambda* parameter can be used in the Dependency attribute of an adjustable variable (see Section 20.4). The same *lambda* parameter can be used in ConvexHullEx in regions of different uncertain parameters to define a dependency between the uncertain parameters. As the *lambda* parameter is not an ordinary uncertainty parameter, it cannot be used in uncertainty constraints.

More flexibility

Through the Uncertainty attribute of an uncertain parameter you can define a relation in term of other ordinary and uncertain parameters in your model which must hold for the uncertain value of that parameter.

The Uncertainty attribute

Consider the following declaration

Example

```
Parameter Demand {
    IndexDomain  : (c,t);
    Property     : Uncertain;
    Uncertainty  : Demand.level(c,t) + Sum[k, D(c,t,k) * xi(k)];
}
```

where D(c,t,k) is an ordinary parameter and xi an uncertain parameter. The reference to Demand.level in the Uncertainty attribute refers to the deterministic (or nominal) value of Demand. The uncertain value of Demand is defined as its nominal value plus a linear combination of some other uncertain parameter xi(k).

Note that the Region and Uncertainty attributes are non-exclusive, i.e., you can use them in conjuction to each other. In such a case, AIMMS will make sure that the solution is robust with respect to both relations.

Non-exclusive attributes

The Region and the Uncertainty attribute of a uncertain parameter can be used to specify possible realizations of the uncertain parameters. In some cases, however, more flexibility is needed in specifying special relations for one or more uncertain parameters. For this purpose AIMMS allows you to specify UncertaintyConstraints. An UncertaintyConstraint is a constraint that specifies the relation between uncertain parameters. It is similar to an ordinary constraint in which the uncertain parameters play the role for variables; the

Uncertainty constraints

definition of an `UncertaintyConstraint` may only refer to normal and uncertain parameters, and not to variables.

The following example specifies a condition on an uncertain parameter that cannot be expressed through its `Region` or `Uncertainty` attributes.

Example

```
Parameter A {
    IndexDomain  : (i,j);
    Property     : Uncertain;
}
UncertaintyConstraint ConditionOnA {
    IndexDomain  : i;
    Definition   : Sum( j, A(i,j) ) <= 1;
}
```

Through the `Constraint` attribute of an `UncertaintyConstraint` you can specify to which (normal) constraints the `UncertaintyConstraint` should apply. In this way it is possible to use different uncertainty sets for different constraints. If the `Constraints` attribute is empty then the `UncertaintyConstraint` will be active for all constraints.

The Constraints attribute

Consider the following declarations

Example

```
UncertaintyConstraint ConditionOnA {
    IndexDomain  : i;
    Constraints  : CapacityRestriction(j) : UncertaintyDependency(i,j);
    Definition   : Sum( j, A(i,j) ) <= 1;
}
Constraint CapacityRestriction {
    IndexDomain  : j;
    Definition   : Sum( i, A(i,j) * Transport(i,j) ) <= Capacity(j);
}
Parameter UncertaintyDependency {
    IndexDomain  : (i,j);
    Definition   : 1 $ (i = j);
}
```

These declarations yield that the uncertainty constraint `ConditionOnA(i)` is only active for constraint `CapacityRestriction(j)` for all elements `j` equal to `i`.

Besides linear uncertainty constraints, AIMMS also allows you to formulate the following uncertainty set for a uncertain parameter ξ, that generalizes the ellipsoidal uncertainty sets that can be defined by using the `Ellipsoid` region:

Generalized ellipsoid

$$\xi^T Q_0 \xi + \sum_{m=1}^{M} \sqrt{\xi^T Q_m \xi} \le b,$$

where Q_0 and Q_m should be positive semidefinite matrices. If your model contains an ellipsoidal uncertainty constraint then the robust counterpart will become a second-order cone program, except if the ellipsoidal uncertainty constraints are of the form

$$\sum_i \sqrt{\xi_i^2} \le b,$$

in which case the robust counterpart will be a linear program.

20.3 Chance constraints

In the previous sections we assumed that each constraint with uncertain data was satisfied with probability 1. In many situations, however, such a require-ment may lead to solutions that over-emphasize the worst-case. In such cases, it is more natural to require that a candidate solution has to satisfy a con-straint with uncertain data for "nearly all" realizations of the uncertain data. More specifically, in this approach one requires that the robust solution has to satisfy the constraint with probability at least $1 - \epsilon$, where $\epsilon \in [0, 1]$ is a pre-specified small tolerance. Instead of the deterministic constraint

Chance constraints

$$a^T x \le b$$

we now require that the *chance constraint*

$$\text{Prob}\left(x : a(\xi)^T x \le b\right) \ge 1 - \epsilon$$

be satisfied, where the probability is associated with the specific distribution of the uncertain parameter(s) ξ.

In general, (linear) problems with chance constraints are very hard to solve even if the probability distribution of the uncertain data is completely known. It is, however, possible to construct safe tractable approximations of chance constraints using robust optimization (see, for instance, Chapter 2 of [BT09]). The way a chance constraint is approximated depends merely on the general characteristics of the data distribution, rather than on precise specification of the distribution. If more information is available about the distribution, this will generally result in a tighter approximation. A tighter approximation, how-ever, could result in a more difficult solution process (for instance, requiring second-order cone programming instead of just linear programming).

Approximation

The procedure to introduce chance constraints into your robust optimization model is as follows:

Chance constraints in Aimms

- indicate which parameters in your model should become random, and specify the properties of their distributions, and
- specify which constraints should be considered chance constraints, and specify their probability and method of approximation.

A probability distribution is modeled in AIMMS as a numeric Parameter for which the Random property has been set (see also Section 4.1). If the property Random is set, AIMMS will create the mandatory Distribution attribute for this parameter which must be used to specify the characteristics of the distribution to be used for that parameter. All random parameters for which a distribution has been specified are considered to be independent.

Random parameters

Consider the following declaration

Example

```
Parameter Demand {
    IndexDomain  : i;
    Property     : Random;
    Distribution : Bounded(Demand(i).level,0.1);
}
```

This declaration states that parameter Demand corresponds to a bounded probability distribution with a mean equal to the nominal value of Demand and a support of 0.1.

AIMMS supports the distribution types listed in Table 20.1 All distributions

Supported distributions

Distribution	Meaning
Bounded(m,s)	mean m with range $[m-s, m+s]$
Bounded(m,l,u)	range $[l,u]$ and mean m not in the center of the range
Bounded(m_l, m_u, l, u)	range $[l,u]$ and mean in interval $[m_l, m_u]$
Bounded(m_l, m_u, l, u, v)	range $[l,u]$ and mean in interval $[m_l, m_u]$, and variance bounded by v
Unimodal(c,s)	unimodal around c with range $[c-s, c+s]$
Symmetric(c,s)	symmetric around c with range $[c-s, c+s]$
Symmetric(c,s,v)	symmetric around c with range $[c-s, c+s]$, and variance bounded by v
Support(l,u)	range $[l,u]$ (and no information about the mean)
Gaussian(m_l, m_u, v)	Gaussian with mean in interval $[m_l, m_u]$ and variance bounded by v

Table 20.1: Supported distribution types for robust optimization

in this table are bounded except the Gaussian distribution. The distributions Bounded(m_l, m_u, l, u), Bounded(m_l, m_u, l, u, v) and Symmetric(c,s,v) are currently not implemented.

A distribution is called unimodal if its density function is monotonically increasing up to a certain point c and monotonically decreasing afterwards. For symmetric distribution AIMMS offers the possibility to mark it as unimodal by using the `unimodal` keyword:

Symmetric unimodal distribution

```
Parameter Demand {
    IndexDomain  : i;
    Property     : Random;
    Distribution : Symmetric(Demand(i).level,0.1), unimodal;
}
```

The `unimodal` keyword can only be used in combination with a symmetric distribution.

In addition to specifying a random parameter using an independent distribution, AIMMS also allows you to define a random parameter as a linear combination of other random parameters (but not as combination of uncertain parameters). For example,

Linear relation

```
Parameter Demand {
    Property     : Random;
    Distribution : Sum( i, xi(i) );
}
```

where `xi` is an random parameter. To avoid cyclic definitions, AIMMS requires that the distributions of random parameters cannot be specified as an expression of other random parameters which are themselves defined as an expression of random parameters.

A constraint in your mathematical program becomes a chance constraint in the context of robust optimization by setting its `Chance` property. The definition of a chance constraint may only contain random parameters, normal parameters and variables. Uncertain parameters are not allowed inside a chance constraint. When setting the `Chance` property for a constraint, you must specify two new attributes for the constraint, the `Probability` attribute and the `Approximation` attribute. It is allowed to use chance constraints in a mixed-integer program.

Chance constraints

The `Probability` attribute specifies the probability with which the chance constraint should be satisfied when solving a robust optimization model. The value of the `Probability` attribute should be a numerical expression in the range $[0, 1]$. If the probability is 0, then AIMMS will not generate the chance constraint. If the probability is 1, then AIMMS will generate an uncertainty constraint.

The Probability attribute

The `Approximation` attribute is used to define the approximation that should be used to approximate the chance constraint. Its value should be an element expression into the predefined set `AllChanceApproximationTypes`.

The Approximation attribute

The approximations supported by AIMMS are:

Supported approximation types

- *Ball*,
- *Box*,
- *Ball-box*,
- *Budgeted*, and
- *Automatic*.

A detailed mathematical definition of these approximation types can be found in Chapter 2 of [BT09]. Whether or not a particular approximation type is possible, depends on the characteristics of the distributions used in the chance constraint, as explained below. By specifying approximation type *Automatic* the most accurate approximation possible will be used. In some cases it might be beneficial to use a less tight approximation because it leads to a robust counterpart that is easier to solve.

Consider the declaration

Example

```
Constraint ChanceConstraint {
    IndexDomain   : i;
    Property      : Chance;
    Definition    : Demand(i) * X(i) <= 10;
    Probability   : prob(i);
    Approximation : 'Ball';
}
```

This declaration states that `ChanceConstraint` is a chance constraint with probability `prob(i)`, and that approximation type *Ball* is used to approximate the chance constraint.

Table 20.2 shows for each (supported) distribution which approximation types are possible. It also shows whether the approximation will result in a linear or a second-order cone robust counterpart. For the `Bounded`(m,s) distribu-

Possible approximations per distribution

Distribution	Automatic	Ball	Box	Ball-box	Budgeted
`Bounded`(m,s)	linear	conic	linear	conic	linear
`Bounded`(m,l,u)	conic		linear		
`Unimodal`(c,s)	conic		linear		
`Symmetric`(c,s) (unimodal)	conic	conic	linear	conic	linear
`Support`(l,u)	linear		linear		
`Gaussian`(m_l,m_u,v)	conic				

Table 20.2: Allowed approximations and their resulting problem type

tion the automatic approximation equals the *Budgeted* approximation, and the automatic approximation of the Support(l, u) distribution equals the *Box* approximation. The non-unimodal Symmetric(c, s) distribution is treated as a Bounded(m, s) distribution.

A chance constraint cannot contain both bounded random parameters and Gaussian random parameters. Different types of bounded random parameters can be combined, in which case only a part of the available information will be used. The possible combinations of bounded random parameters are given in Table 20.3.

Combining distributions

	Distribution	1	2	3	4	5
1	Bounded(m, s)	1	2	–	1	5
2	Bounded(m, l, u)	2	2	–	2	5
3	Unimodal(c, s)	–	–	3	3	5
4	Symmetric(c, s) (unimodal)	1	2	3	4	5
5	Support(l, u)	5	5	5	5	5

Table 20.3: Resulting distribution type when combining distributions

If a random parameter with a Bounded(m, l, u) distribution and a random parameter with a Support(l, u) distribution are used in a single chance constraint, then Table 20.3 states that the Bounded(m, l, u)) distribution of the first random parameter will be treated as a Support(l, u) distribution. Unimodal distributions can only be mixed with unimodal Symmetric(c, s) and Support(l, u) distributions.

Explanation

20.4 Adjustable variables

An *adjustable variable* reflects a decision made after uncertain data has been revealed. In robust optimization this is interpreted as the adjustable variable taking some (explicit or implicit) functional form in terms of the uncertain data on which it depends. In AIMMS, you indicate that a Variable should be treated as adjustable by setting its Adjustable property.

Adjustable variables

For any adjustable variable, AIMMS will create a Dependency attribute which you can use to specify on which uncertain parameters the variable depends. The attribute value must be a comma-separated list of mappings from an uncertain parameter to a binary parameter, indicating for which combination of indices a dependency exists on that uncertain parameter.

The Dependency attribute

AIMMS currently only supports the *linear decision rule*, which means any adjustable variable will be expressed as an affine relation in terms of the uncertain parameters which it depends on. More explicitly, if an adjustable variable $x(t)$ depends on uncertain parameters d_r, then, under the linear decision rule, AIMMS assumes that $x(t)$ takes the form

Linear decision rule only

$$x(t) = X_0(t) + \sum_r X_r(t)d_r$$

where $X_0(t)$ and $X_r(t)$ are newly introduced intermediate variables, the value of which is determined by solving the robust counterpart. As such, the value of an adjustable variable is not fully determined by the solver. It can be computed afterwards for a given realization of the uncertain parameters. AIMMS will automatically generate the affine relation based on the dependencies you indicated in the Dependency attribute, without the need for you to introduce the appropriate intermediate variables.

In order for AIMMS to be able to generate the robust counterpart of a robust optimization model, the model must satisfy the *fixed recourse* condition, i.e., the coefficients of any adjustable variables in your model must not depend on uncertain parameters. In addition, for AIMMS to be able to generate the robust counterpart, adjustable variables may *not* occur in chance constraints. Also, adjustable variables cannot be integer.

Requirements for adjustable variables

The collection of intermediate variables introduced during this process, automatically becomes available through the .Adjustable attribute of the adjustable variable at hand, followed by the name of the uncertain parameter involved. That is, if an adjustable variable x(i) depends on an uncertain parameter a(j), then the corresponding intermediate variable is available as the expression x.Adjustable.a(i,j). In addition, a variable x.Adjustable.Constant(i) will be created to account for the constant part of the affine relation. If necessary, you can bound these variables through the .Lower and .Upper suffices, or you can formulate additional constraints on these variables.

The .Adjustable suffix for variables

Consider the following declarations

Example

```
Variable Stock {
    IndexDomain  :  t;
    Property     :  Adjustable;
    Dependency   :  Demand(t2) : StockDemandDependency(t,t2);
}
Parameter Demand {
    IndexDomain  :  t;
    Property     :  Uncertain;
}
Parameter StockDemandDependency {
    IndexDomain  :  (t,t2);
    Definition   :  1 $ (t2 < t);
}
```

These declarations yield that the adjustable variable Stock(t) depends on the uncertain parameter Demand(t2) for all elements t2 smaller than t. Given these declaration, AIMMS will generate the following definition for Stock(t)

```
Stock(t) = Stock.Adjustable.Constant(t) +
        sum(t2 | StockDemandDependency(t,t2), Stock.Adjustable.Demand(t,t2)*Demand(t2))
```

If the data for Demand(t) becomes available, you can use the computed values of Stock.Adjustable.Demand(t,t2) and Stock.Adjustable.Constant to compute the value of Stock(t).

To compute the values of an adjustable variable for a given realization of the uncertain parameters of the robust optimization model, you do not have to explicitly add the appropriate definitions to your model. AIMMS offers the function GMP::RobustEvaluateAdjustableVariables, discussed in Section 16.8, to automatically compute these values for you.

Evaluating adjustable variables

20.5 Solving robust optimization models

After you have specified all uncertain parameters, random parameters, chance constraints and adjustable variables that specify your robust optimization model, your original mathematical program can now be solved as a robust optimization model. It is also still possible to solve it as a *deterministic* model by just calling the SOLVE statement (see also Section 15.3).

Solving robust optimization models

To solve a robust optimization model for a MathematicalProgram *MP*, the first step is to generate its robust counterpart. This can be accomplished by calling the GMP function

Generate robust counterpart

- GenerateRobustCounterpart(*MP*,*UncertainParameters*, *UncertaintyConstraints*[,*Name*])

The function returns an element into the set AllGeneratedMathematicalPrograms, i.e., the generated mathematical program representing the robust counterpart of the given robust optimization model.

Through the *UncertainParameters* and *UncertaintyConstraints* arguments you can specify the collection of uncertain and random parameters, as well as the uncertainty constraints that you want to take into account when generating the robust counterpart. Together, these completely determine the uncertain data which AIMMS will use to translate the uncertain matrix coefficients, chance constraints and adjustable variables into the generated mathematical program representing the robust counterpart.

Specifying uncertain data

With the optional *Name* argument you can explicitly specify a name for the generated mathematical program. If you do not choose a name, AIMMS will use the name of the underlying `MathematicalProgram` as the name of the generated mathematical program as well. Please note, that AIMMS will also use this name as the default name for solving the deterministic model. Therefore, if you do not want the generated mathematical program of the deterministic model to be deleted, then you have to choose a non-default name.

Name argument

You can solve the generated mathematical program *gmp* representing the robust counterpart by calling the regular GMP procedure

Solving the robust counterpart

- `GMP::Instance::Solve(`*gmp*`)`

The `GMP::Instance::Solve` method is discussed in full detail in Section 16.2. Alternatively, you can use any of the other available functions available to solve generated mathematical programs discussed in Chapter 16. Note that AIMMS will not allow you to use the GMP modification functions on any `gmp` generated by `GenerateRobustCounterpart`.

The solution resulting from solving the robust counterpart will satisfy all non-chance constraints in your model for all realizations of the uncertain parameters that you passed to the `GenerateRobustCounterPart` function, and will satisfy all chance constraints with the given probabilities and approximations, given the random parameters taken into account.

The resulting solution

Bibliography

[BT09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*, Prince-
        ton University Press, Princeton, N.J., 2009.