How to Contact Dash

USA, Canada and all Americas
Dash Optimization Inc

Information and Sales: info@dashoptimization.com
Licensing: license-usa@dashoptimization.com
Product Support: support-usa@dashoptimization.com
Tel: +1 (201) 567 9445
Fax: +1 (201) 567 9443
Dash Optimization Inc.
560 Sylvan Avenue
Englewood Cliffs
NJ 07632
USA

Japan
Dash Optimization Japan

Information and Sales: info@jp.dashoptimization.com
Licensing: license@jp.dashoptimization.com
Product Support: support@jp.dashoptimization.com
Tel: +81 43 297 8836
Fax: +81 43 297 8827
WBG Marive-East 21F FASuC B2124
2-6 Nakase Mihama-ku
261-7121 Chiba
Japan

Worldwide
Dash Optimization Ltd

Information and Sales: info@dashoptimization.com
Licensing: license@dashoptimization.com
Product Support: support@dashoptimization.com
Tel: +44 1926 315862
Fax: +44 1926 315854
Leam House, 64 Trinity Street
Leamington Spa
Warwickshire CV32 5YN
UK

For the latest news and Xpress-MP software and documentation updates, please visit the Xpress-MP website at http://www.dashoptimization.com or subscribe to our mailing list.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notation</td>
<td>1</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>2</td>
</tr>
<tr>
<td><strong>I Stochastic Programming with mmsp</strong></td>
<td>3</td>
</tr>
<tr>
<td>2 Stochastic Programming (SP) basics</td>
<td>4</td>
</tr>
<tr>
<td>2.1 2-stage stochastic problems</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Multi stage stochastic problems</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Scenario generation</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Underlying deterministic model</td>
<td>7</td>
</tr>
<tr>
<td>2.5 Expanding the underlying deterministic model</td>
<td>7</td>
</tr>
<tr>
<td>2.5.1 Node based</td>
<td>7</td>
</tr>
<tr>
<td>2.5.2 Scenario based</td>
<td>8</td>
</tr>
<tr>
<td>2.5.2.1 Non-anticipative constraints (NAC)</td>
<td>9</td>
</tr>
<tr>
<td>2.6 Solution measures</td>
<td>9</td>
</tr>
<tr>
<td>2.6.1 Recourse problem (R.)</td>
<td>9</td>
</tr>
<tr>
<td>2.6.2 Expected Value problem</td>
<td>10</td>
</tr>
<tr>
<td>2.6.2.1 Expected Value problem with recourse</td>
<td>10</td>
</tr>
<tr>
<td>2.6.3 Perfect Information problem</td>
<td>11</td>
</tr>
<tr>
<td>2.6.3.1 Perfect information problem with recourse</td>
<td>11</td>
</tr>
<tr>
<td>2.6.4 Focus on an instance of the problem</td>
<td>12</td>
</tr>
<tr>
<td><strong>3 Xpress-SP architecture</strong></td>
<td>13</td>
</tr>
<tr>
<td>3.1 Scope</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Xpress-SP module (mmsp)</td>
<td>13</td>
</tr>
<tr>
<td>3.2.1 Types</td>
<td>13</td>
</tr>
<tr>
<td>3.2.2 Functions and Procedures</td>
<td>14</td>
</tr>
<tr>
<td>3.2.3 Parameters</td>
<td>14</td>
</tr>
<tr>
<td>3.2.4 Constants</td>
<td>14</td>
</tr>
<tr>
<td>3.3 IVE controls and views for SP</td>
<td>14</td>
</tr>
<tr>
<td><strong>4 Building stochastic models</strong></td>
<td>15</td>
</tr>
<tr>
<td>4.1 Setting stages</td>
<td>15</td>
</tr>
<tr>
<td>4.2 SPPrands and SPPrandexps</td>
<td>15</td>
</tr>
<tr>
<td>4.2.1 Declaring and associating sprand with stages</td>
<td>15</td>
</tr>
<tr>
<td>4.2.2 Operators</td>
<td>16</td>
</tr>
<tr>
<td>4.2.3 Relational conditions</td>
<td>16</td>
</tr>
<tr>
<td>4.2.4 Logical conditions</td>
<td>16</td>
</tr>
<tr>
<td>4.2.5 Functions</td>
<td>16</td>
</tr>
<tr>
<td>4.2.5.1 One argument</td>
<td>16</td>
</tr>
<tr>
<td>4.2.5.2 Two arguments</td>
<td>16</td>
</tr>
<tr>
<td>4.2.5.3 Three arguments</td>
<td>16</td>
</tr>
<tr>
<td>4.3 Scenario tree</td>
<td>17</td>
</tr>
<tr>
<td>4.3.1 Generation</td>
<td>17</td>
</tr>
<tr>
<td>4.3.1.1 Exhaustive tree</td>
<td>17</td>
</tr>
<tr>
<td>4.3.1.2 Stage symmetric</td>
<td>20</td>
</tr>
</tbody>
</table>
Notation

$T$: Number of stages (indexed by $t, t', t''$)

$S$: Number of scenarios (indexed by $s$)

$P_s$: Unconditional probability of occurrence of $s^{th}$ scenario

$\xi_t$: Random event in $t^{th}$ stage

$N_t$: Number of nodes in $t^{th}$ stage of the scenario tree (indexed by $n$)

$\text{node}(t, n)$: $n^{th}$ node in $t^{th}$ stage

$\xi_{tn}$: $n^{th}$ realized values of $\xi_t$ at $\text{node}(t, n)$

$p_{tn}$: Conditional probability of occurrence of $n^{th}$ realization $\xi_t$ at $\text{node}(t, n)$

$P_{tn}$: Unconditional probability of occurrence of $n^{th}$ realization of $\xi_t$ at $\text{node}(t, n)$
Chapter 1

Introduction

*Stochastic Programming (SP)* is a special class of mathematical programming that involves optimization under uncertainty. Such problems occur in various streams of industry, e.g., supply chain management, financial optimization, network optimization, energy sector, etc. The Xpress stochastic programming suite provides a range of tools and functionalities for modeling, solving, and analyzing these problems.

We begin by understanding the basics of SP in Chapter 2, where we study what attributes of SP problems distinguish them from deterministic problems. Chapter 3 describes the design of the Xpress-SP framework. There we look at various devices embedded in the Xpress-SP suite for an efficient and rapid modeling of SP problems and other tools for solving and analyzing them. Chapter 4 gives a detailed description of these tools and functionalities required to model and solve an SP problem in this framework. Finally, Chapter 5 illustrates how stochastic models in various fields can be built using the Xpress-SP suite.
I. Stochastic Programming with *mmsp*
Stochastic Programming (SP) basics

Stochastic problems can essentially be divided into 2-stage and multi-stage problems. In a 2-stage problem, the initial decisions are taken first. These are then followed by a random event. Next, the recourse decisions, which are based on this random event, are taken. The multi-stage problem, as the name suggests, consists of multiple stages, with a random event occurring between each stage. In this case, decisions are made at each stage. The following sections give a detailed description of these problems:

2.1 2-stage stochastic problems

In a 2-stage problem, first, the initial or the first stage decisions (e.g., system design decisions) are made which are followed by random events such as demand, availability, price, or a combination of these. Then the second stage decisions (e.g., operational decisions) are made. This can be viewed pictorially as follows:

In the standard form, the 2-stage stochastic program is:

\[
\begin{align*}
\text{optimize} & \quad c_1^T x_1 + E_2 \left[ \tilde{c}_2^T (\xi_2) x_2 (\xi_2) \right] \\
\text{s.t} & \quad A_{11} x_1 \approx b_1 \\
& \quad \tilde{A}_{21} (\xi_2) x_1 + \tilde{A}_{22} (\xi_2) x_2 (\xi_2) \approx \tilde{b}_2 (\xi_2) \quad \forall \xi_2 \in \Xi_2 \\
& \quad l_1 \leq x_1 \leq u_1 \\
& \quad \tilde{l}_2 (\xi_2) \leq x_2 (\xi_2) \leq \tilde{u}_2 (\xi_2) \quad \forall \xi_2 \in \Xi_2
\end{align*}
\]

where \( \approx \) indicates \( \leq, \geq, = \) or combination of these. \( x_1 \) and \( x_2 \) are the first and the second stage stochastic decision variables respectively. \( E_2[] \) is the expectation operator with respect to

![Figure 2.1: 2-stage problem](image)
the random event $\xi_2$. $\Xi_2$ represents the state space (the set of possible outcomes or the values that $\xi_2$ can assume) in the recourse stage. There may be separate values for the objective coefficients $c_2(\xi_2)$, right hand side $b_2(\xi_2)$, and matrix coefficients $A_{21}(\xi_2)$ and $A_{22}(\xi_2)$ for each outcome $\xi_2$ in $\Xi_2$, and the recourse decisions $x_2(\xi_2)$ depend on $\xi_2$, i.e., there are separate sets of recourse decisions for each outcome $\xi_2$ in the second stage.

### 2.2 Multi stage stochastic problems

Multi-stage problems can be viewed pictorially as follows:

Formally, the multi-stage stochastic program is defined as follows:

\[
\text{optimize } \bar{c}_1'x_1 + E_2[\bar{c}_2'x_2 + E_3[\bar{c}_3'x_3 + E_4[....+ E_T[\bar{c}_T'x_T]]]]
\]

\[
s.t
\]

\[
\sum_{t=1}^{T} \bar{A}_{tt'}x_t \approx \bar{b}_{t'} \quad \forall t' \in \{1, ..., T\}
\]

\[
\bar{l}_t \leq x_t \leq \bar{u}_t \quad \forall t \in \{1, ..., T\}
\]

where

- ‘optimize’ indicates maximize or minimize
- ‘\bar{'}’ indicates random data variable
- \(\approx\) indicates \(\leq, \geq, =\) or a combination of these

\[
x_t = x_t(\xi_2, \xi_3, ... , \xi_T) \quad \forall t \in \{1, ..., T\}
\]

\[
\bar{c}_t' = \bar{c}_t'(\xi_2, \xi_3, ... , \xi_t) \quad \forall t \in \{1, ..., T\}
\]

\[
\bar{b}_{t'} = \bar{b}_{t'}(\xi_2, \xi_3, ... , \xi_{t'}) \quad \forall t' \in \{1, ..., T\}
\]

\[
\bar{A}_{tt'} = \bar{A}_{tt'}(\xi_2, \xi_3, ... , \xi_{t'}, \xi_{t''}) \quad \forall t \in \{1, ..., t'\}, t' \in \{1, ..., T\}
\]

\[
\bar{l}_t = \bar{l}_t(\xi_2, \xi_3, ... , \xi_T) \quad \forall t \in \{1, ..., T\}
\]

\[
\bar{u}_t = \bar{u}_t(\xi_2, \xi_3, ... , \xi_T) \quad \forall t \in \{1, ..., T\}
\]

\[
E_t[.] = E_{\xi_t}(\xi_2, \xi_3, ... , \xi_{t-1})[.\] \quad \forall t \in \{2, ..., T\}
\]

Each of the above entities depends on the sequence of events $(\xi_2, \xi_3, ...)$.

---

1Note that $c_1$, $A_{11}$, $b_1$, $l_1$, and $u_1$ are not random.
2.3 Scenario generation

ξ_t being a random variable, takes different values with different probabilities. If each of the ξ_t’s can be discretized, the realized values of ξ_t give rise to what is called a scenario tree. An example of a scenario tree with T (number of stages) = 3 and S (number of scenarios) = 4 is shown in Figure 2.3.

In the t^{th} stage there are N_t nodes. For each stage t in 2 to T, each ξ_t has N_t outcomes (ξ_{t1} w. p. p_{t1}, ξ_{t2} w. p. p_{t2}, . . . , ξ_{tN_t} w. p. p_{tN_t}), where p_{tn} is the conditional probability of visiting the n^{th} node in the t^{th} stage from its parent node in the t−1^{th} stage. The realizations of ξ_2, ξ_3, . . . , ξ_T correspond to scenarios (paths in the tree). For each stage t and each node n in that stage, the node has an unconditional probability P_{tn} of being visited, which is equal to the product of conditional probabilities along the path to that node. Similarly, each scenario s has a scenario probability P_s that is equal to the product of conditional probabilities along the path to that scenario. The following observations can be made about the scenario tree:
1. \[ \sum_{n':\text{Node}(t+1,n') \in \text{Children}(	ext{Node}(t,n))} p_{t+1,n'} = 1 \quad \forall n \in \{1,\ldots,N_t\}, t \in \{1,\ldots,T-1\} \]

2. \[ P_m = \prod_{(t',n'):\text{Node}(t',n') \in \text{path to Node}(t,n)} p_{m} \quad \forall n \in \{1,\ldots,N_t\}, t \in \{2,\ldots,T\} \]

3. \[ \sum_{n=1}^{N_t} P_m = 1 \quad \forall t \in \{2,\ldots,T\} \]

4. \[ P_s = \prod_{(t,n):\text{Node}(t,n) \in \text{path to scenario } s} p_{m} \quad \forall s \in \{1,\ldots,S\} \]

5. \[ \sum_{s=1}^{S} P_s = 1 \]

2.4 Underlying deterministic model

If we ignored the randomness of the data momentarily, then an underlying deterministic model can be written as follows:

\[
\begin{align*}
\text{optimize} \quad & \sum_{i=1}^{T} \tilde{c}_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{T} \tilde{A}_{i,i} x_i \approx \tilde{b}_i, \quad \forall t' \in \{1,\ldots,T\} \\
& \tilde{l}_i \leq x_i \leq \tilde{u}_i, \quad \forall t \in \{1,\ldots,T\}
\end{align*}
\]

2.5 Expanding the underlying deterministic model

Given the dependency of the coefficients \((\tilde{c}_i, \tilde{A}_{i,i}, \tilde{b}_i, \tilde{l}_i, \tilde{u}_i)\) of a stochastic program on the random events \((\xi_i)\), it can automatically be expanded into an extensive form (deterministic equivalent problem) by introducing new variables and constraints. There are basically two ways of creating new variables and constraints: node based and scenario based.

2.5.1 Node based

Given a scenario tree, the underlying deterministic model can be expanded into an extensive mathematical program based on the nodes. The basic idea is to add a subscript of a node number to each of the stochastic decision variables \((x_i)\) becomes \(x_{t,n}\) for \(n = 1, \ldots, N_t\). Then the resulting extensive deterministic model would be:
In this model, \( c_{tn} \), \( A_{tn} \), \( b_{tn} \), \( l_{tn} \), \( u_{tn} \) are the resolved values of \( \tilde{c}_{t} \), \( \tilde{A}_{t} \), \( \tilde{b}_{t} \), \( \tilde{l}_{t} \), \( \tilde{u}_{t} \) at the node \((t, n)\).

Based on above formulation, the matrix structure of the problem shown in Figure 2.3 would look as follows:

2.5.2 Scenario based

A stochastic model can also be expanded based on the scenarios. Here each variable is also subscripted by scenarios \((x_{t} \rightarrow x_{t,s} \text{ for } s = 1,...,S)\). The expanded mathematical program would look as follows:

**optimize** \[ \sum_{t=1}^{T} \sum_{n=1}^{N_s} P_{tn} c'_{tn} x_{tn} \]

s.t.

\[ \sum_{(t', n') : \text{Path to Node}(t, n)} A_{t'n'} x_{tn} \approx b_{t'n'} \quad \forall n' \in \{1, ..., N_{t'}\}, t' \in \{1, ..., T\} \]

\[ l_{tn} \leq x_{tn} \leq u_{tn} \quad \forall n \in \{1, ..., N_{t}\}, t \in \{1, ..., T\} \]

In this model \( c_{tn} \), \( A_{tn} \), \( b_{tn} \), \( l_{tn} \), \( u_{tn} \) are the resolved values of \( \tilde{c}_{t} \), \( \tilde{A}_{t} \), \( \tilde{b}_{t} \), \( \tilde{l}_{t} \), \( \tilde{u}_{t} \) at the node \((t, n)\). Based on above formulation, the matrix structure of the problem shown in Figure 2.3 would look as follows:
2.5.2.1 Non-anticipative constraints (NAC)

Consider the node(2, 2) in Figure 2.3. When the model is parsed according to scenarios, although we have two separate variables $x_{22}$ and $x_{23}$ corresponding to scenarios 2 and 3 at this node, both of them should assume same value since they cannot depend on the future events ($x_t \equiv x_t(\xi_s, \ldots, \xi_t)$). Therefore, the following set of non-anticipative constraints also needs to be included in the above formulation:

$x_{ts} = x_{ts'} \forall s, s' \in \Lambda_{tn}$, where $\Lambda_{tn}$ is the set of scenarios passing through the node(t, n). Hence, the matrix structure for the problem shown in Figure 2.3 would look as follows.

2.6 Solution measures

2.6.1 Recourse problem (R.)

The standard stochastic problem discussed in Section 2.2 is also referred to as the recourse
2.6.2 Expected Value problem

If the $\xi_i$'s are replaced by their expected values in the recourse problem, then such a problem is called an expected value (EV) problem.

\[
\begin{align*}
\mathbf{z_{EV}} & = \text{optimize } \sum_{t=1}^{T} \tilde{c}_t' x_t \\
\text{sv} & \\
\sum_{t=1}^{T} \tilde{A}_{t,t} x_t & \approx \tilde{b}_{t'} & \forall t' \in \{1,...,T\} \\
\tilde{l}_t & \leq x_t \leq \tilde{u}_t & \forall t \in \{1,...,T\}
\end{align*}
\]

where,

\[
\begin{align*}
\tilde{c}_t & = \tilde{c}_t(\overline{\xi}_2, \overline{\xi}_3, ..., \overline{\xi}_T) & \forall t \in \{1,...,T\} \\
\tilde{b}_{t'} & = \tilde{b}_{t'}(\overline{\xi}_2, \overline{\xi}_3, ..., \overline{\xi}_{t'}) & \forall t' \in \{1,...,T\} \\
\tilde{A}_{t,t'} & = \tilde{A}_{t,t'}(\overline{\xi}_2, \overline{\xi}_3, ..., \overline{\xi}_{t'}) & \forall t \in \{1,...,t'\}, t' \in \{1,...,T\} \\
\tilde{l}_t & = \tilde{l}_t(\overline{\xi}_2, \overline{\xi}_3, ..., \overline{\xi}_T) & \forall t \in \{1,...,T\} \\
\tilde{u}_t & = \tilde{u}_t(\overline{\xi}_2, \overline{\xi}_3, ..., \overline{\xi}_T) & \forall t \in \{1,...,T\} \\
\tilde{\xi}_t & = E[\xi_t | \xi_2 = \overline{\xi}_2, ..., \xi_{t'-1} = \overline{\xi}_{t'-1}] & \forall t \in \{2,...,T\}
\end{align*}
\]

2.6.2.1 Expected Value problem with recourse

Let $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_T)$ be the optimal solution to the above problem, then for each stage an expected recourse problem can be defined as follows.
\[ Z_{EV_t}(t) = \sum_{t'=1}^{t} \tilde{c}_{t'}^i \tilde{x}_i + \text{optimize} \left[ \sum_{t'=t+1}^{T} \tilde{c}_{t'}^i x_{t+1}^i + E_{t+2}[\ldots + E_T(\tilde{c}_{T}^i x_T)] \right] \]

Subject to:
\[ \sum_{t'=t+1}^{T} \tilde{A}_{t'} \tilde{x}_i - \sum_{t'=1}^{t} \tilde{A}_{t'} \tilde{x}_i \geq \forall t' \in \{t+1, \ldots, T\} \]
\[ \tilde{l}_i \leq x_i \leq \tilde{u}_i \quad \forall t' \in \{t+1, \ldots, T\} \]

Here, the first \( t \) stage variables are fixed at their optimal values obtained in the EV problem.

The following observations can be made:

1. \( Z_{EV} \) is not worse than \( Z_R \) (i.e., if maximizing, \( Z_{EV} \geq Z_R \)).
2. If \( \sum_{t'=1}^{t} \tilde{A}_{t'} \tilde{x}_i \approx b_{t'} \), \( \tilde{l}_i \leq \tilde{x}_i \leq \tilde{u}_i \), \( \forall t' \in \{1, \ldots, t\} \) and \( EV^t \) is feasible then, \( Z_R \) is not worse than \( Z_{EV^t}(t) \).
3. If 2. is true then the value of stochastic solution: \( VSS(t) \) is the absolute value of the difference between \( Z_R \) and \( Z_{EV^t}(t) \), else it is \( +\infty \).

### 2.6.3 Perfect Information problem

Here, the problem for each scenario is solved independently.

\[ Z_{PI}(s) = \text{optimize} \sum_{t=1}^{T} c_{t}^s x_t^s \]

Subject to:
\[ \sum_{t'=1}^{t} A_{t'} x_{t'}^s \approx b_{t'} \quad \forall t' \in \{1, \ldots, T\} \]
\[ l_{t} \leq x_{t}^s \leq u_{t} \quad \forall t \in \{1, \ldots, T\} \]

Let \( x^*_s = (x_{t_1}^s, x_{t_2}^s, \ldots, x_{t_p}^s) \) be the optimal solution to the above problem, \( x^* = \sum_{s=1}^{S} P_s x^*_s = (x_{t_1}, x_{t_2}, \ldots, x_{t}) \) be the aggregated solution, and \( Z_{PI} = \sum_{s=1}^{S} P_s Z_{PI}(s) \) be the aggregated objective value.

#### 2.6.3.1 Perfect information problem with recourse

\[ Z_{PIr}(t) = \sum_{t=1}^{t} c_{t}^s x_t^* + \text{optimize} \left[ \sum_{t'=t+1}^{T} c_{t'}^s x_{t+1}^* + E_{t+2}[\ldots + E_T(\tilde{c}_{T}^i x_T)] \right] \]

Subject to:
\[ \sum_{t'=t+1}^{T} \tilde{A}_{t'} x_{t'}^* - \sum_{t'=1}^{t} \tilde{A}_{t'} x_{t'}^* \geq \forall t' \in \{t+1, \ldots, T\} \]
\[ \tilde{l}_i \leq x_i \leq \tilde{u}_i \quad \forall t' \in \{t+1, \ldots, T\} \]
The following observations can be made:

1. \( Z_{P} \) is not worse than \( Z_{RP} \).

2. If \( \sum_{t'} t A_{tt'} x_{t'} \approx b_{t}, \quad l_{t} \leq x_{t'} \leq u_{t'} \forall t' \in \{1, \ldots, t\} \) and \( P_{tr} \) is feasible then, \( Z_{P} \) is not worse than \( Z_{P_{tr}}(t) \).

3. If 2. is true then the expected value of perfect information: \( EV_{PI} \) is the absolute value of the difference between \( Z_{R} \) and \( Z_{P_{tr}} \), else it is \( +\infty \).

2.6.4 Focus on an instance of the problem

Instead of looking at the fully expanded problem at once, one could also look at the instance of the underlying problem in a particular scenario or at a node in a scenario tree. The corresponding

model in a scenario \( s \)

\[
\begin{align*}
\text{optimize} & \quad \sum_{t=1}^{T} c_{t} x_{t} \\
\text{s.t.} & \quad \sum_{t=1}^{T} A_{t t'} x_{t} \approx b_{t}, \quad t' \in \{1, \ldots, t\} \\
& \quad l_{t} \leq x_{t} \leq u_{t} \quad t \in \{1, \ldots, T\}
\end{align*}
\]

model at a node \((t, n)\) in the scenario tree

\[
\begin{align*}
\text{optimize} & \quad \sum_{(t', n'):\text{node}(t', n') \text{ to Path to node}(t, n)} c_{t' n'} x_{t' n'} \\
\text{s.t.} & \quad \sum_{(t', n'):\text{node}(t', n') \text{ to Path to node}(t', n')} A_{t' n'} x_{t' n'} \approx b_{t}, \quad \forall (t', n') : \text{node}(t', n') \in \text{Path to node}(t, n) \\
& \quad l_{t' n'} \leq x_{t' n'} \leq u_{t' n'} \quad \forall (t', n') : \text{node}(t', n') \in \text{Path to node}(t, n)
\end{align*}
\]

e.g. in Figure 2.3 the model focused at node \((3, 4)\) will be:

\[
\begin{align*}
\text{optimize} & \quad c_{1,1} x_{1,1} + c_{2,3} x_{2,3} + c_{3,4} x_{3,4} \\
\text{s.t.} & \quad A_{1,1} x_{1,1} \approx b_{1,1} \\
& \quad A_{2,1} x_{1,1} + A_{2,2} x_{2,3} \approx b_{2,3} \\
& \quad A_{3,1} x_{1,1} + A_{3,2} x_{2,3} + A_{3,3} x_{3,4} \approx b_{3,4} \\
& \quad l_{1,1} \leq x_{1,1} \leq u_{1,1} \\
& \quad l_{2,3} \leq x_{2,3} \leq u_{2,3} \\
& \quad l_{3,4} \leq x_{3,4} \leq u_{3,4}
\end{align*}
\]
Chapter 3
Xpress-SP architecture

This chapter describes the organization of SP tools in Xpress. First, we discuss the scope of the current version of Xpress-SP. This is followed by a brief description of the SP module, where all the functionality for modeling stochastic models is defined and implemented. Finally, the controls and functionalities available in Xpress-IVE for visualization and analysis purposes are also described.

3.1 Scope

Xpress-SP is focused on the ease of modeling a stochastic problem. Specifically, available functionalities allow users to create a scenario tree in a structured and flexible fashion. The redundancy in modeling stochastic problems (vs. deterministic problems) is minimized, e.g., elimination of writing non-anticipative constraints, expanding the problem to retain the matrix ordered by stage or scenarios (see Section 2.5), etc. Similarly, tools are provided in IVE to manipulate scenarios (Sections 4.3.1.3, 4.7), visualize the values of entities in the model (Section 4.5), and analyze the solution (Section 4.5).

This version does not support any specialized algorithms (see Appendix) to solve stochastic problems; however, users can use the standard algorithms like simplex (Primal/Dual), Barrier, and Branch & Bound techniques available in the XpressMP suite for solving stochastic problems. Users may also obtain instance of the problem in a scenario or at a node in a scenario tree and devise their own algorithm to solve the problem. Although no special simulation or statistical tools are provided for scenario generation, users can easily create a scenario tree if one or more random variables are distributed independently or a set of random variables are distributed jointly with known discretized values and probabilities. Other functionalities may also be used to create scenario trees if the discretized values and probabilities of random variables are known.

3.2 Xpress-SP module (mmsp)

Xpress stochastic programming functionalities are made available in a module called ‘mmsp’, so users must use this module in their stochastic models. The design of this module is based on different types defined for stochastic model entities and different functions for building, solving, and analyzing these models as discussed below.

3.2.1 Types

The following new types are defined:

- **sprand**: random variable that takes different values with certain probability, e.g., demand data.
- **sprandexp**: random expression built using real and sprand.
• **spvar**: stochastic decision variable that takes different values in different scenarios or at different nodes in the scenario tree.

• **splinctr**: Stochastic constraint built with real, sprand, sprandexp, and spvar.

### 3.2.2 Functions and Procedures

The following functions and procedures are provided:

- abs, ceil, exp, floor, getdual, getdualatnode, getdualinscen, getobjval, getcost, getcostatnode, getcostinscen, getslack, getslackatnode, getslackinscen, getsol, getsolatnode, getsolsonscen, ln, maximize, minimum, minimize, minimum, round, spaddchildren, spaggregate, spcreatetree, spdelscenario, speval, spfix, spgenexhtree, spgentree, spgetchild, spgetchildrencount, spgetnodecount, spgetparent, spgetprobcond, spgetprobbscens, spgetprobuncond, spgetscencount, spgetxprsrealattrib, spgetxprsintattrib, spgetxprsstrattrib, spif, spprinttree, spsetdist, spsethidden, spsetname, spsetprobcond, spsetprobcondatnode, spsetprobscen, spsetrandatnode, spsetstage, spsetstages, spsettype, spunfix, spwriteprob

### 3.2.3 Parameters

The following control parameters are defined:

- xsp_cblog, xsp_ive_enable, xsp_ftable, xsp_tolprob_lb, xsp_tolprob_ub,
  xsp_tolprob_totlb, xsp_tolprob_totub, xsp_cond_prob, xsp_scen_based,
  xsp_xtprs_prob, xsp_implicit_stage, xsp_verbose, xsp_scale_probs,
  xsp_disp_warnings, xsp_re_init, xsp_create_tree, xsp_loadnames

### 3.2.4 Constants

The following constants are defined:

- XSP_X_SP, XSP_X_SMPS, XSP_SMPS_FREE, XSP_X_LP, XSP_STAGE_ZERO, XSP_REC,
  XSP_PRIMAL, XSP_PI, XSP_OPT_LP_ONLY, XSP_EV, XSP_DUAL, XSP_BARRIER, XSP_GLOBAL.

- XSP_MIN, XSP_MAX, XSP_LP_OPTIMAL, XSP_LP_INFEAS, XSP_LP_CUTOFF, XSP_LP_UNFINISHED,
  XSP_LP_UNBOUNDED, XSP_LP_CUTOFF_IN_DUAL, XSP_MIP_NOT_LOADED, XSP_MIP_LP_NOT_OPTIMAL,
  XSP_MIP_LP_OPTIMAL, XSP_MIP_NO_SOL_FOUND, XSP_MIP_SOLUTION, XSP_MIP_INFEAS, XSP_MIP_OPTIMAL

### 3.3 IVE controls and views for SP

The following entities can be viewed in IVE:

- Underlying expanded matrix
- Scenario tree
  - List view
  - Pie view
  - Block view
- Stochastic dashboard
  - Model information
  - Realized values of stochastic entities with respect to scenarios
  - Distribution of stochastic entities
  - Scenario manipulation (aggregation and deletion)
Chapter 4

Building stochastic models

In this chapter we look in detail at how users can build stochastic models in Xpress-SP using the functionalities available in the *mmsp* module. The steps and issues related to scenario generation, model formulation, and solution analysis are addressed here. Although the framework for building stochastic models is flexible, users are advised to follow the model-building procedures in a sequential fashion as shown below.

### 4.1 Setting stages

Stages are set using the function `spsetstages(Stages: set);` e.g.,

```
declarations
T=4
Stages=1..T
end-declarations
spsetstages(Stages)
```

The set `Stages` can be of type `integer` or `string`, or it can be a range.

### 4.2 SPrands and SPrandexps

Each `sprand` corresponds to a stochastic event (\(\xi_t\)), and hence its stage and possible values must be set by the user. `sprandexps` on the other hand are expressions built using `sprand`, and therefore their stages and values are automatically defined.

#### 4.2.1 Declaring and associating sprand with stages

Random variables are declared as `sprand` in *mmsp*.

```
declarations
  demand, price :array(2..T) of sprand
end-declarations
```

It is recommended that at the time of declaration of arrays of `sprand`, all the sets used for indexing such arrays be constant or finalized (i.e. elements of sets are known and sizes of sets are fixed). At the time of declaration, `sprand` are automatically ‘created’, however if one or more of the sets used for indexing an array of `sprand` are dynamic then by default such an array is `dynamic` in nature. In this case, users will have to explicitly create all the `sprand` using the `create` procedure (e.g., `T:=4;forall(t in 2..T) create(demand(t))`.

Users can set the stage of `sprand` using the function `spsetstage(srnd: sprand). One or more sprand in each stage may correspond to the \(\xi_t\) for that stage; e.g.,
forall(t in 2..T) spsetstage(demand(t),t)
forall(t in 2..T) spsetstage(price(t),t)

In the above example, the pair \((\text{demand}_t, \text{price}_t)\) corresponds to the random event \(\xi_t\). Note that users may not have to set the stages of all \text{sprand} if the parameter \text{xsp_implict_stage} is set to \text{true} (see Section 4.5). Also note that one could dissociate an \text{sprand} from a stage by setting its stage to \text{XSP_STAGE_ZERO}.

4.2.2 Operators

The operators: +, -, *, /, and ^ can be used with types \text{real}, \text{sprand}, and \text{sprandexp} to create new \text{spradexp}, e.g.,

\[
\text{declarations}
\quad \text{tot_demand: sprandexp}
\quad \text{end-declarations}
\quad \text{tot_demand:=sum(t in 2..T) demand(t)}
\]

4.2.3 Relational conditions

Relational conditions, which themselves are of type \text{sprandexp}, can be created using the relational operators: \(<\leq, \geq, and =\), together with \text{real}, \text{sprand}, and \text{sprandexp}.

4.2.4 Logical conditions

Logical conditions, which in turn are of type \text{sprandexp} themselves, can be created using relational conditions together with ‘and’, ‘or’, ‘not’ or a combination of these, e.g.,

\[
\text{declarations}
\quad \text{if_demand_demand: sprandexp}
\quad \text{end-declarations}
\quad \text{if_pos_demand:= (demand(2)+demand(3))}\geq\text{demand(1)} \text{ or}
\quad \quad \text{demand(1)}>0
\]

4.2.5 Functions

The general format of all the functions involving \text{sprand} and \text{sprandexp} is: \(f(\text{args}):\text{sre}\) where \(f\) is the name of the function, \(\text{args}\) are the arguments of this functions, and \(\text{sre}\) is a \text{sprandexp}. This is further described in the following section.

4.2.5.1 One argument

\(f\): abs, ceil, floor, round, ln, exp
arg1: \text{sprand} or \text{sprandexp}

4.2.5.2 Two arguments

\(f\): maximum, minimum
arg1, arg2: \text{real}, \text{sprand}, or \text{sprandexp}

4.2.5.3 Three arguments

\(f\): spif
arg1: \text{sprandexp} (must be a logical or relational condition)
arg2, arg3: \text{real}, \text{sprand}, or \text{sprandexp} (arg2, arg3 correspond to true and false values of the condition)
e.g., if $sr_1$, $sr_2$, and $sr_3$ are sprand then following is a valid usage of spif:

$sre:=\text{spif}(sr_1 \leq 3 \text{ and } sr_1 \geq sr_2 \text{sr}_3, 1, \text{maximum}(sr_2, \text{ceil}(sr_1)))$

Here whenever $arg_1$ evaluates to true, $sre$ is assigned $arg_2$ otherwise it is assigned $arg_3$.

### 4.3 Scenario tree

#### 4.3.1 Generation

The first step in building a stochastic model is the creation of the scenario tree (see e.g. Section 2.3).

where,

- $T =$ Number of stages
- $S =$ Number of Scenarios
- $N_t =$ Number of nodes in stage $t$ ($N_1 = 1, N_T = S$)
- $P_s =$ Probability of scenario $s$, such that $\sum_s P_s = 1$
- $\xi_{t,n} =$ Assumed value of random variable $\xi_t$ at node $(t, n_t)$
- $P_{t,n} =$ Conditional probability of occurrence of event $\xi_{t,n}$ (at node $(t, n_t)$)

Creation of a scenario tree involves setting of its stages followed by declaration and association of each sprand with a stage.

#### 4.3.1.1 Exhaustive tree

*All independently distributed sprand*

If all the random variables in the model are independent, then one can first specify their dis-
cretized distribution and generate the tree as follows:

- Assume \( T=4 \)
- Distribution of demand 2 is \( \{5 \text{ w.p. } 0.5, 10 \text{ w.p. } 0.5\} \) in the second stage
- Distribution of demand 3 is \( \{7 \text{ w.p. } 0.2, 12 \text{ w.p. } 0.6, 13 \text{ w.p. } 0.2\} \) in the third stage
- Distribution of demand 4 is \( \{12 \text{ w.p. } 0.6, 14 \text{ w.p. } 0.4\} \) in the fourth stage

\[
\begin{align*}
declarations 
nElem=3 
Elems = 1..nElem 
demand:array(2..T) of sprand 
value,probability:array(2..T,Elems) of real 
val,prob:array(Elems) of real 
end-declarations 

value:=\{5,10,0, 
7,12,13, 
12,14,0\} 
probability:=\{.5,.5,0, 
.2,.6,.2, 
.6,.4,0\} 

forall(t in 2..T) do 
forall(e in Elems) prob(e):=0 
forall(e in Elems) prob(e):=probability(t,e) 
forall(e in Elems) val(e):=value(t,e) 
spsetdist(demand(t),val,prob) 
end-do 
spgenexhtree 
\]

This will create a tree with 2 branches from first stage nodes, 3 branches from second stage nodes, and 2 branches from third stage nodes, implying a total of \( 2 \cdot 3 \cdot 2 = 12 \) scenarios.

Note that in the exhaustive generation, the size of the scenario tree increases rapidly with an increase in the number of sprand. If in the above-mentioned example, we add another random variable price, with price 2, price 3, and price 4 assuming 3, 2, and 2 discrete values in the second, third, and fourth stage respectively, then the number of scenarios would be \( (2 \cdot 3) \cdot (3 \cdot 2) \cdot (2 \cdot 2) = 144. \)

Some jointly distributed sprand

One can also create a scenario tree with some or all sprand jointly distributed. The following example demonstrates how to build a 3-stage scenario tree, with three sprand namely \( sr_1, sr_2 \) and \( sr_3 \), where \( sr_1 \) and \( sr_2 \) belong to the second stage and \( sr_3 \) belongs to the third stage. It is assumed that \( sr_1 \) and \( sr_3 \) are jointly distributed, whereas \( sr_2 \) is independently distributed.

\[
\begin{align*}
declarations 
T=3 
Stage=1..T 
sr1,sr2,sr3: sprand 
end-declarations 

! set stages 
spsetstages(Stage) 
spsetstage(sr1,2) 
spsetstage(sr2,2) 
spsetstage(sr3,3) 

! set independent distribution of sr2 = \{10 \text{ w.p. } .2, 20 \text{ w.p. } .8\} 
declarations 
ind_val,ind_prob:array(1..2) of real 
end-declarations 

ind_val:=\{10,20\} 
ind_prob:=\{.2,.8\} 
spsetdist(sr2,ind_val,ind_prob) 
\]

Figure 4.2: Exhaustive scenario tree
Figure 4.3: Exhaustive scenario tree with some jointly distributed random variables

![Exhaustive scenario tree diagram]

The following exhaustive tree is created:

4.3.1.2 Stage symmetric

Creating trees

Users may also want to create symmetric trees, e.g., a tree with three branches from each node in each stage. This can be done using the function

```
spcreatetree(3)
```

Users must specify the realized values and the conditional or the scenario probabilities afterwards. Similarly, a tree with two branches from the first stage node, 3 branches from each node in the second stage, and 2 branches from each node in the third stage, can be generated as follows:
Figure 4.4: Explicit scenario tree - node numbers

```plaintext
declarations
  branches:array(2..T) of integer
end-declarations
branches:=[2,3,2]
spcreatetree(branches)
```

**Specifying values of sprand and conditional probabilities**

Once a tree is created explicitly, the default value of all sprand in the scenario tree is zero. mmsp provides functions for setting the assumed value of sprand in the scenario tree based on the tree’s structure. If the tree is symmetric with respect to stage (or is symmetric across all the stages), then it can be accomplished as follows:

```plaintext
! Assume branches:=[2,3,2] is already set
declarations
  MaxBranch=3
  Branches=1..MaxBranch
  value:array(2..T,Branches) of real
  Val:array(Branches) of real
  probability:array(2..T,Branches) of real
  prob:array(Branches) of real
end-declarations

value:=[5,10,0,
    7,12,13,
    12,14,0]
forall(t in 2..T) do
  forall(b in Branches) Val(b):=value(t,b)
  spsetrand(demand(t),Val)
end-do

probability:=[.5,.5,0,
             .2,.6,.2,
             .6,.4,0]
forall(t in 2..T) do
  forall(b in Branches) prob(b):=probability(t,b)
  spsetcondprob(t,prob)
end-do
```

This will generate a scenario tree as shown in Figure 4.2.

### 4.3.1.3 Explicit tree

Users may also create a tree (usually asymmetrical) using the following functionalities in mmsp:

**Creating a tree**

Assume $T=3$, and user wants to create the following tree

- In one shot:
This can be achieved as follows:

```plaintext
declarations
T=3        ! Number of stages
S=3        ! Number of scenarios
NodeNum:array(1..S,1..T) of integer
end-declarations

NodeNum:=[1,1,1,
         1,1,2,
         1,2,3]
spcreatetree(NodeNum)
```

Here, the array `NodeNum` is created by sequentially entering the node number across the stages for each scenario; e.g., in the second scenario, the node numbers corresponding to the first, second and third stage are 1, 1 and 2 respectively, hence the second row of `NodeNum` in the above example is [1, 1, 2].

- **Node by node:**

Alternatively, the tree can be created as follows:

```plaintext
spaddchildren(1,1,2)
spaddchildren(2,1,2)
spaddchildren(2,2,1)
```

Here, the function `spaddchildren(t, n, b)` adds `b` children nodes at node(t, n).

### Creating a tree with trap stages

A tree with a trap stage consists of at least one path that does not necessarily end at the last stage. An example of such a tree is shown in the following figure:

### Setting realized sprand and probabilities

An `sprand` can be set using the function `spsetrandatnode`, e.g.,
The scenario tree thus created would look as follows:

For the scenario-based problems, users may set the scenario probabilities explicitly using the function `spsetprobscen(scen_num: integer, prob: real)`.

### 4.3.2 Manipulation

Xpress-SP supports a few scenario manipulation routines such as aggregation and deletion of scenarios. Scenario manipulation can also be done interactively in Xpress-IVE (see Section 4.7).

#### 4.3.2.1 Aggregating all scenarios emerging from a common ancestral node

All the scenarios emerging from an ancestral node in a scenario tree can be aggregated to create one aggregated scenario. Consider following scenario tree:
Here, scenarios 1 to 3 can be aggregated as follows:

\texttt{spaggregate(1..3)}

Note that the new scenario tree is renumbered as follows:

Hence, in order to aggregate scenarios 1 to 3 and scenarios 4 to 5 shown in Figure 4.10 separately, the function call should be made in the following order:

\texttt{spaggregate(4..5)}
\texttt{spaggregate(1..3)}

This will result in the following tree:

Similarly, scenarios 1 to 5 can be aggregated to create a one-scenario scenario tree.

**Aggregated probabilities**

If scenarios 1 to 2 were aggregated in the above shown tree, the new tree with aggregated values and probabilities would look as follows:

Here, \(7 \cdot .2 + 12 \cdot .8 = 11\) with probability 1.0. The probability of the aggregated scenarios is the sum of probabilities of the scenarios aggregated.

### 4.3.2.2 Aggregating selected scenarios emerging from a common ancestral node

One may also aggregate a set of scenarios (not necessarily contiguous), if none of the nodes after the common ancestral node of the selected scenarios in the path pertaining to the selected scenarios have any siblings. This is illustrated in the following example:
Figure 4.10: Original tree

Figure 4.11: Aggregated values

Figure 4.12: 4-stage scenario tree
In the scenario tree shown above, scenarios 1 and 5 can be aggregated since the node(3, 1), node(3, 3), node(4, 1), and node(4, 5) lying after the common ancestral node(2, 1) do not have any siblings. The corresponding function call in mmsp would be spaggregate({1,5}).

On aggregation, the nodes corresponding to the selected scenarios are deleted. The new aggregated branch corresponding to the aggregated scenario is appended as the last child of the common ancestral node, and the tree node numbers are updated suitably. The aggregated scenario tree in the above example would look as follows:

4.3.2.3 Deletion in scenario based problems

Users can delete a scenario using the function spdelscen(s:integer). In this case, the probabilities of the remaining scenarios are re-normalized (i.e., each remaining scenario’s probability is divided by one minus probability of deleted scenario). Similarly a set of scenarios can be deleted using the function spdelscen(delSet :set of integer). Again, the probabilities are re-normalized (i.e., each remaining scenario’s probability is divided by one minus the sum of probability of deleted scenarios), and based on the updated scenario probabilities, the conditional probabilities are re-normalized. The sprand remain unchanged, whereas the node numbers are updated suitably. This is illustrated pictorially in the following example.

If scenarios 3 and 5 are deleted using the function spdelscen({3,5}), the resulting scenario tree would look as follows:

4.3.2.4 Deletion in node based problems

Deletion in node-based problems is similar to the one in scenario-based problems, with the exception that the re-normalization of the probabilities is done based on the conditional probabilities of the nodes associated with the deleted scenarios. The following example demonstrates deletion of set of scenarios in a node based problem:

If scenarios 3 and 5 are deleted using spdelscen({3,5}), then the resulting scenario tree looks as follows:
Figure 4.14: 3-stage scenario tree - scenario based

Figure 4.15: Pruned 3-stage tree - scenario based
Figure 4.16: 3-stage scenario tree - node based

Figure 4.17: Pruned 3-stage tree - node based
4.3.3 Traversing

Users may use several scenario tree related functions such as:

- `spgetscencount`: to get total number of scenarios ($S$)
- `spgetnodecount(t)`: to get total number of nodes at stage $t$ ($N_t$)
- `spgetparent(t,n)`: to get the node number of the parent of $node(t, n)$
- `spgetchildrencount(t,n)`: to get total number of children nodes of $node(t, n)$
- `spgetchild(t,n,b)`: to get node number of $b^{th}$ child of $node(t, n)$
- `spgetprobscen(s)`: to get unconditional probability of $s^{th}$ scenario ($P_s$)
- `spgetprobcond(t,n)`: to get conditional probability of $node(t, n)$ ($P_{tn}$)
- `spgetprobuncond(t,n)`: to get unconditional probability of $node(t, n)$ ($P_{tn}$)

4.3.4 Evaluating

One can evaluate the realized value of `sprand` and `sprandexps` in the scenario tree using the function `speval(sptype, scen):real` or `speval(sptype, stage, node):real` e.g., if $sr1$ and $sr2$ are of type `sprand`, and $sre$ is an `sprandexp`, then the value of $sre:=sr1+sr2$ at the fourth node in the third stage can be evaluated by calling the function `speval(sre,3,4)`. Calling `speval(sre)` would return the expected value of $sre$.

4.3.5 Debugging

One can print the scenario tree loaded in Xpress-SP by calling the function `spprinttree`.

4.3.6 Formulation

Formulating a stochastic problem in the Xpress-SP syntax involves declarations of `spvar` and `splinctr`. The model formulation is discussed below.

4.3.7 Stochastic variables and constraints

Each stochastic variable must be associated with a stage. The stage of a constraint is automatically defined by the stages of the variables and other stochastic entities used to build the constraint.

4.3.7.1 Declaring and associating `spvar` with stages

The `spvar` are declared as follows:

```plaintext
declarations
x:array(1..T) of spvar
end-declarations
```

It is recommended that at the time of declaration of arrays of `spvar`, all the sets used for indexing such arrays are ‘finalized’ (i.e. elements of sets are known and sizes of sets are fixed). At the time of declaration, `spvar` are automatically 'created', however if one or more of the sets used for indexing an array of `spvar` are not ‘finalized’ (e.g. in the above example, if the value of $T$ is not known at the time of declaration), then by default such an array is ‘dynamic’ in nature. In this case, users will have to explicitly ‘create’ all the `spvar` using the procedure `create` (e.g., $T:=4$; forall(t in 1..T) create(x(t))).

`Spvars` may be associated with a stage (unless `xsp_implict_stage` is set to true (see Section 4.5)) e.g.
forall(t in 1..T) spsetstage(x(t),t)

One could also disassociate a spvar from a stage by setting its stage to XSP_STAGE_ZERO.

4.3.7.2 Setting types and bounds

The type of decision variables can be set using the is_integer, is_binary, and is_free attributes. The bounds can be set by using unnamed unary constraints, e.g., if x is of type spvar and srn is of type sprandexp, then the constraint x<=srn sets the upper-bound of x. Of course, the bounds can be real, sprand or sprandexp.

4.3.7.3 Writing the objective function and constraints

The objective function and constraints (type splinctr) may be declared as follows:

```plaintext
declarations
    Obj, Ctr:splinctr
end-declarations
```

The following points must be kept in mind while a writing model in Xpress-SP:

- The objective function must be unconstrained (i.e., without a ≤, ≥, or =).
- An splinctr has an implied stage, which is the maximum of the highest stage of sprand and the highest stage of spvar in that splinctr.
- Users may set the type of a splinctr as special ordered set (is_sos1, is_sos2) or as a global constraint (spsettype(Ctr, XSP_GLOBAL)) that chains terms across different realizations in a scenario tree weighted by probabilities of realizations. Special ordered set type constraints cannot contain ‘fixed’ variables (see Section 4.3.7.3), and global constraints cannot occur in expected value (see Section 2.6.2) or focused (see Section 2.6.4) problems.
- In a node based problem, for each spvar in the objective function, the stage of the sprand in its coefficient must not be greater than the stage of the spvar itself (see Section 2.2).

4.3.8 Manipulation

spvar may be fixed at a certain value using the function spfix(x: spvar, val: real). spunfix can be used on spvar to unfix them. Similarly, splinctr may be hidden or unhidden using the function spsethidden(c: splinctr, cond: boolean). If the second is true, then the constraint is hidden, else it is unhidden. Note that if all the variables of a constraint are fixed, then the constraint is automatically hidden. These functionalities are particularly useful if the user wants to fix some variables e.g., the entire first stage variables at a predetermined value and solve the problem in the recoursed stochastic framework (see Section 2.6).

4.4 Stochastic problem

The stochastic problem is automatically generated when optimization routine (maximize or minimize) is called. The type of the matrix generated depends on whether the problem is node based or scenario based. Additionally, related problems may also be generated by passing suitable options to the optimization routine.
4.4.1 Setting Xpress optimizer control variables

Users may set the optimizer control parameters using the function `setparam` e.g., `setparam('xprs_presolve', 0)` will turn the presolve off. All the controls related to the optimizer are listed in the `mmsp` module.

4.4.2 Optimizing

The problem can be optimized by calling the procedures `maximize(Obj: splinctr)` or `minimize(Obj: splinctr)`. At this point, the validity of the scenario tree is checked. If no tree exists then an exhaustive tree is attempted to be built (by calling the function `spgenexh` internally). The problem is then internally expanded and sent to the optimizer. The optimizer control parameters for this problem are also set.

4.4.2.1 Instance of problem in a scenario

Instead of optimizing the fully expanded problem, one could optimize an instance of a problem in a particular scenario s (see Section 2.6.4) by calling `maximize(Obj: splinctr, s: integer)` or `maximize(Obj: splinctr, s: integer)`. Note that the parameter `xsp_scen_based` should be set to `true`.

4.4.2.2 Instance of problem at a node in the scenario tree

Similarly one could optimize an instance of a problem at a particular node $(t,n)$ (see Section 2.6.4) in a scenario tree by calling `maximize(Obj: splinctr, t: integer, n: integer)` or `maximize(Obj: splinctr, t: integer, n: integer)`. Note that the parameter `xsp_scen_based` should be set to `false`.

4.4.2.3 Options

Users may also pass an option string to indicate the algorithm and type of problem to be solved by calling the function `maximize(Obj: splinctr, option: string)` or `minimize(Obj: splinctr, option: string)`. The options and their meanings are listed below:

- `XSP_OPT_LP_ONLY`: solve LP only
- `XSP_PRIMAL`: use primal simplex
- `XSP_DUAL`: use dual simplex
- `XSP_BARRIER`: use barrier
- `XSP_REC`: solve recourse problem
- `XSP_EV`: solve expected value problem (see Section 2.6.2)
- `XSP_PI`: solve perfect information problem (see Section 2.6.3)
- `XSP_REC+XSP_EV`: solve recoursed expected value problem by fixing first stage variables and hiding first stage constraints (see Section 2.6.2.1).
- `XSP_REC+XSP_PI`: solve recoursed perfect information problem by fixing first stage variables and hiding the first stage constraints (see Section 2.6.3.1).

As an example, if the user wants to maximize ‘Profit’ — an `splinctr` — by solving only the LP relaxation of a 4-stage mixed integer problem in a recoursed expected value framework by fixing first stage variables at their expected values and hiding the first stage constraints, using Xpress’ barrier routine, the corresponding function call would be:

`maximize(Profit, XSP_OPT_LP_ONLY+XSP_BARRIER+XSP_EV+XSP_REC)`. 
4.4.3 Exporting problems

Using the optimization routines spexportprob, maximize or minimize and the constants defined in Xpress-SP one could also export problems. The spexportprob procedure is overloaded as follows:

- \text{spexportprob}(\text{dir}, \text{objfn}, \text{filetype})
- \text{spexportprob}(\text{dir}, \text{objfn}, \text{filetype}, \text{filename})

where argument,

1. \text{dir} can be \text{XSP\_MAX} or \text{XSP\_MIN}
2. \text{objfn} is the objective function of type \text{splinctr}
3. \text{filetype} can be \text{XSP\_X\_LP}, or \text{XSP\_X\_SP}, or \text{XSP\_X\_SMPS}, or \text{XSP\_X\_SMPS\_FREE}
4. \text{filename} is the name of the file in which the exported problem will be stored.

4.4.3.1 SP file

For the purpose of debugging the underlying SP model, one could export the problem .sp file. For instance, \text{spexportprob}($\text{XSP\_MIN}$, \text{Obj}, \text{XSP\_X\_SP}, ‘myfile’) will export the problem in the file \text{myfileProb.sp}, whereas \text{minimize}($\text{Obj}$, \text{XSP\_X\_SP}) will both export the stochastic problem in the file \text{ObjProb.sp} as well as minimize the problem.

4.4.3.2 SMPS files

The time, core, and scenario files pertaining to stochastic problem can be exported e.g., in the free format by calling \text{exportprob}($\text{XSP\_MAX}$, \text{obj}, \text{XSP\_X\_SMPS\_FREE}). This will create the files \text{objSMSP.tim}, \text{objSMSP.sto}, and \text{objSMPS.cor}. Alternatively user may create these files and optimize the problem by calling \text{maximize}($\text{obj}$, \text{XSP\_X\_SMPS\_FREE}). Note that if the constant \text{XSP\_X\_SMPS} is used then files are exported in the fixed format.

4.4.3.3 LP problem

The problem currently loaded in the optimizer can be exported in an lp file by calling \text{spexportprob}($\text{XSP\_MAX}$, \text{obj}, \text{XSP\_X\_LP}) or \text{maximize}($\text{obj}$, \text{XSP\_X\_LP}).

4.4.4 Getting attributes of a problem from the optimizer

If problem is solved using Xpress optimizer, users may get the status of the problem after optimization by obtaining various integer, real, and string attributes using the function \text{getparam}. It is advised that the user checks the optimization status, e.g., \text{xprs\_lpstatus} and/or \text{xprs\_ mipstatus}, before getting the values of the objective function and other related solutions, comparing the values of these attributes with the constants — \text{XSP\_LP\_OPTIMAL}, \text{XSP\_LP\_INFEAS}, \text{XSP\_LP\_CUTOFF}, \text{XSP\_LP\_UNFINISHED}, \text{XSP\_LP\_UNBOUNDED}, \text{XSP\_LP\_CUTOFF\_IN\_DUAL}, \text{XSP\_MIP\_NOT\_LOADED}, \text{XSP\_MIP\_LP\_NOT\_OPTIMAL}, \text{XSP\_MIP\_LP\_OPTIMAL}, \text{XSP\_MIP\_NO\_SOL\_FOUND}, \text{XSP\_MIP\_SOLUTION}, \text{XSP\_MIP\_INFEAS}, or \text{XSP\_MIP\_OPTIMAL}.

4.4.5 Getting solution

If the problem is feasible then users can obtain the solution value and reduced cost of \text{spvar}, and the activity or slack value of \text{splinctr} in a scenario or at a node in the scenario tree. The values of the first stage entities can be found using the following functions:
These functions may also be used for \texttt{svar} or \texttt{splinctr} belonging to any stage in an Expected value (see Sections 2.6.2.1) or a focused (see Sections 2.6.4) problem.

### 4.4.5.1 Solution in a scenario

For fully expanded problems, the values of entities can also be obtained in a particular scenario, e.g.,

\begin{verbatim}
getsolinscen(svar: spvar, scen: integer):real
getrcostinscen(svar: spvar, scen: integer):real
getdualinscen(sctr: splinctr, scen: integer):real
getslackinscen(sctr: splinctr, scen: integer):real
\end{verbatim}

### 4.4.5.2 Solution at a node in the scenario tree

Similarly, the solution can also be obtained at a particular node:

\begin{verbatim}
getsolatnode(svar: spvar, node: integer):real
getrcostatnode(svar: spvar, node: integer):real
getdualatnode(sctr: splinctr, node: integer):real
getslackatnode(sctr: splinctr, node: integer):real
\end{verbatim}

Note: if \texttt{svar} or \texttt{scctr} belongs to stage $t$, then node must belong to \{1, ..., $N_t$\}.

### 4.4.6 Evaluation

One can evaluate the solution or fixed value of \texttt{svar} or realized value of and \texttt{splinctr} in the scenario tree using the function \texttt{speval}. The function syntax is similar to the one shown in Section 4.7.2.2.

### 4.4.7 Other functions

Users may also override the names of \texttt{svar}, \texttt{splinctr}, and \texttt{sprand} that are displayed in IVE, by setting their names using the procedure \texttt{spsetname}.

### 4.4.8 Debugging

#### 4.4.8.1 Writing the problem loaded in the optimizer memory

The fully expanded problem is loaded in the optimizer only at the point when optimization is called; hence, a problem can be exported only after optimization is complete. The function for exporting the problem is \texttt{spwriteprob(filename: string, flags: string)}. It replicates the function \texttt{XPRSwriteprob} provided in the Xpress optimizer library. Please refer to the optimizer reference manual for description on the parameters ‘filename’ and ‘flags’.

### 4.5 Setting SP parameters

The following control parameters are available in Xpress-SP suite:

- \texttt{XSP\_IVE\_ENABLE}  
  This parameter when set to \texttt{false}, disables IVE access to entities in stochastic model.  
  \textit{Default}: \texttt{true}.  

Building stochastic models 33 Xpress-SP Reference Manual
XSP_TOLPROB_LB, XSP_TOLPROB_UB These are the tolerances on the probabilities (e.g., distribution, conditional, and scenario probabilities). Specifically, if $p$ is one of these probabilities, then $XSP_TOLPROB_LB \leq p \leq 1+XSP_TOLPROB_UB$.
Default: $10^{-6}$, $10^{-6}$ respectively.

XSP_TOLPROB_TOTLB, XSP_TOLPROB_TOTUB These are the tolerances on the total probabilities that should sum to 1 (e.g., distribution, conditional, and scenario probabilities). If $\sum p$ is one these probabilities, then $1-XSP_TOLPROB_TOTLB \leq \sum p \leq 1+XSP_TOLPROB_TOTUB$.
Default: 0.05, 0.05 respectively.

XSP_SCEN_BASED If this is set to true, then the model will be expanded based on scenarios instead of nodes.
Default: 0

XSP_COND_PROB If it is set to false, the conditional probabilities may not sum to 1, and product of the conditional probabilities in a scenario may not be equal to scenario probability. However, if XSP_SCEN_BASED is false, then XSP_COND_PROB is automatically set to true.
Default: true.

XSP_SCALE_PROBS If set to false, the probabilities will not be scaled to sum to 1.
Default: true.

XSP_LOADNAMES If set to false, the names of stochastic entities will not be loaded into matrix.
Default: true.

XSP_IMPLICIT_STAGE Users may not have to set the stages of the sprand and spvar if this parameter is set to true. If the sprand or spvar are grouped in arrays, then they are automatically associated with the stages assuming that the last indexing set is the stage set. If any of the sprand or spvar is not stored in arrays, then it is associated with the first stage. Users may override the automatic association (when XSP_IMPLICIT_STAGE is true) of sprand or spvar with stages by explicitly setting their stage.
Default: false.

XSP_VERBOSE If it is set to false then IVE will disable the optimizer messages that are displayed in the output text window.
Default: true.

XSP_DISP_WARNINGS If it is set to false then warnings will not be displayed.
Default: true.

XSP_RE_INIT May be set to false when solving structurally similar problems sequentially, e.g., problem instances in different scenarios (see Section 2.6.4).
Default: true.

XSP_CREATE_TREE If set to false it prevents creation of nodes in the scenario tree. Helpful, if creation of an exhaustive tree based on distributions is not required. (Note: if xsp_create_tree is set to false then XSP_IVE_ENABLE must be set to false)
Default: true.

### 4.6 Chance constraints

Users may model chance constraints in Xpress-SP by using the global constraints functionality. A regular stochastic constraint, say of the form: $\sum_{t' \in S} x_{t'} \approx \tilde{b}_{t'}$ for some $t'$ is expanded internally as

$$\sum_{(t,n) \in \text{PathToNode}(t',n')} x_{t,n} \leq b_{t',n'}$$

in a node-based problem, i.e., constraints are replicated for all nodes in stage $t'$, or as $\sum_{s=1}^{S} x_{t,s} \approx \tilde{b}_{t,s}$ for all scenarios $s$ in a scenario based problem i.e., constraints are replicated for all scenarios.
By setting such as constraint as of type 'global', it may be expanded as
\[ \sum_{t'=1}^{t'} \sum_{n=1}^{N_t} P_{tn} x_{tn} \approx \sum_{s=1}^{S} P_{s} b_{t's}, \]
where \( b_{t's} \) are the realized values of \( b \) in stage \( t' \) weighted by probabilities of scenarios.

Hence, one may model a chance constraint, for instance in a scenario-based problem, by introducing a binary variable \( z \), and setting a constraint \( C := Z \leq \beta \) of type global by calling
\[ \text{spsettype}(C, \text{XSP\_GLOBAL}). \]
Consequently, the constraint will be expanded internally as
\[ \sum_{s=1}^{S} P_{s} z_{s} \leq \beta, \]
resulting in restricting the total probability of scenarios where \( z \) can be one to \( \beta \).

### 4.7 Analyzing in IVE

The Xpress Integrated Visual Environment (IVE) provides tools for analyzing the matrix structure, scenarios, and the solution values and distributions of various entities in the model across all the scenarios (see Figure 4.18).

IVe provides additional features like SP scenario form, Stochastic Dashboard, and Matrix View for visualizing, manipulating, and analyzing stochastic problems.

#### 4.7.1 SP scenario form

The SP form provides list, pie, and block views of the generated scenario tree.

- The list view lists the scenario indices, their probabilities, and for each stage, the realized values of \( \text{sprand} \) in that stage. Each \( \text{sprand}'s \) name is prefixed by its stage number in curly brackets.
The *pie view* shows the scenario tree in the form of a pie. The first stage forms the innermost circle and the last stage forms the outermost circle. Each stage is colored differently and the nodes in that stage are shaded differently. The size of the node (span of the angle in the circle) is proportional to the cumulative probability of that node (i.e., product of conditional probabilities along the branches to that node). Users can click on a node to see the path to it, its node number, its stage number, the realized values of \( \text{sprand} \), and the conditional and cumulative probability. Users may also double-click on any node and see the realized values of each of the \( \text{sprand} \) along the path to that node. In addition, one can move the pie by clicking and holding the right mouse button and dragging. Users may also zoom in and out by using the centre-wheel of the mouse.

- In the *block view*, the left most block corresponds to the first stage, and the last block corresponds to the last stage. Other features are similar to the ones in the pie-view features except that users can move the tree only vertically (see fig below).

### 4.7.2 Stochastic dashboard

The dashboard provides the model information, namely, the number of stages, \( \text{spvar} \), \( \text{sprand} \), \( \text{splinctr} \), and scenarios. The dashboard also lists the \( \text{sprand} \), \( \text{spvar} \), \( \text{splinctr} \), and scenario indices in the model. Each of these is prefixed by its stage number in curly brackets. The dashboard also gives a miniature version of the scenario tree in the block and the pie view. Users may click on any entity (\( \text{sprand} \), \( \text{spvar} \), or \( \text{splinctr} \)) to see its cumulative probability distribution (with statistics like min, max, median and expected values) across scenarios. Users may also select one or more scenarios with any entity to see its values in those scenarios (see fig below).

One may interactively aggregate and delete scenarios in IVE after scenario generation and
before optimization. The user must check the option *Pause to prune scenario tree manually* in IVE in order to perform these scenario-pruning operations.

### 4.7.2.1 Aggregation

Once the model is running in the *pause mode*, users may click on multiple numbers of nodes to select the scenarios to be aggregated. If the button *Aggregate selected* on the dashboard is pressed, then aggregation is performed automatically and the scenario tree is updated. As mentioned earlier, the new aggregated scenario probability is sum of all the probabilities of the scenarios that were aggregated, and the aggregated sprand are sum of the corresponding conditional–probability weighted sprand.

### 4.7.2.2 Deletion

In the *pause mode*, users may select scenarios to be deleted. One may use the ‘ctrl’ key on the keyboard for multiple selections of scenarios from the scenario list or from the scenario tree, and then press the button *Delete selected*. The remaining scenario and conditional probabilities are re-normalized, and the scenario tree is updated automatically. Note that both of these operations can be performed interactively, and multiple times as long as a valid scenario tree exists. The equivalent mmsp functions are displayed in the text box on the dashboard, which users may copy in their model to replicate the same behavior in the next run. Users may resume execution of the model by pressing the button *Resume execution* on the dashboard.

### 4.7.3 Matrix View

Users may view the underlying expanded matrix in the *Matrix* tab in IVE. The matrix view provides List view to see the solution, and the *Graphical original* and *Graphical presolved* to
Figure 4.21: Aggregation and Deletion in IVE

visualize the original and the pre-solved matrix. The user must select the option for viewing these matrices in IVE. As mentioned in Section 2.5, the matrix can be expanded according to nodes or scenario. In the former case, the variables and constraints are prefixed by their node numbers, whereas in the latter case they are prefixed by their scenario numbers. For the purpose of clarity, these numbers are enclosed in square brackets.

4.7.4 Other standard features

The following standard features of IVE can be used for writing and analyzing the stochastic models:

- Interface for modeling the problem
- Input/Output form for viewing the problem statistics and optimization log
- Branch and bound tree, if the global problem is being solved
Chapter 5
Illustrative examples

In this chapter, we demonstrate the use of Xpress-SP’s modeling and solution techniques in solving stochastic problems from various streams such as supply chain management, finance, energy sector, transportation, and forest planning. A short description of a problem from each sector is given, followed by the model used to solve the problem and its implementation with Mosel.

5.1 Farm production planning

In this section, we show how to model a simple 2-stage model using Xpress-SP. We model the farmer’s problem (see [BL97a]), where a farmer needs to make decisions regarding the devotion of available land to different products: wheat, corn and sugar beet, during the winter season. The yield is realized in the summer and depends on the overall weather during the year (e.g., bad, ok, good). The farmer must produce a certain amount of products for the cattle feed. Then the farmer must also decide how much of each product needs to be purchased or sold depending on the yield. There is an additional restriction on the selling price of sugar beet because of quota restriction. The model entities are described below.

- Sets
  - Stages: {Winter, Summer}
  - Products: {Wheat, Corn, Sugar Beet}

- Constraints
  - Minimum production for cattle feed
  - Quota restriction on sugar beet production

- Variables
  - \( x_i \): acres of land devoted to product \( i \)
  - \( w_{\text{wheat}}, w_{\text{corn}} \): tons of wheat and corn sold respectively
  - \( y_{\text{wheat}}, y_{\text{corn}} \): tons of wheat and corn purchased respectively
  - \( w_{\text{favor}}, w_{\text{unfavor}} \): tons of sugar beet sold at favorable and unfavorable price respectively

5.1.1 Model

```plaintext
model farmer
uses 'mmsp'

parameters
Explicit=false ! Explicitly create tree or use the joint ! distribution feature
```
forward procedure Analyse(prob: string)
setparam('xsp_scen_based', false)
declarations
!
"Wheat","Corn","Sugar Beet Favorable","Sugar Beet Unfavorable"
AllProducts="Wht","Crn","SugBtFav","SugBtUnfav"
Products="Wht","Crn","SugBt"
WheatAndCorn="Wht","Crn"
PlantingCost: array(Products) of real
SellingPrice: array(AllProducts) of real
PurchasePrice: array(WheatAndCorn) of real
MinimumRequirement: array(WheatAndCorn) of real
TotalAvailableLand: real
MaxSugarBeetQuota: real
end-declarations
PlantingCost:= [150,230,260]
SellingPrice:= [170,150,36,10]
PurchasePrice:= [238,210]
MinimumRequirement:= [200,240]
TotalAvailableLand:= 500
MaxSugarBeetQuota:= 6000
declarations
Stages="Wint","Summ"
yield: array(Products) of sprand
x: array(Products) of spvar
y: array(WheatAndCorn) of spvar
w: array(AllProducts) of spvar
TotalCost: splinctr
LandUtilized: splinctr
MinCattleFeedRequirement: array(WheatAndCorn) of splinctr
SugarBeetProduction: splinctr
SugarBeetQuota: splinctr
end-declarations
!
Stages
spsetstages(Stages)
forall(p in Products) spsetstage(yield(p),"Summ")
!
Scenarios
declarations
S = 3
Values: array(Products,1..S) of real
end-declarations
forall(s in 1..S) prob(s):=1/3
Values:= [3,2,5,2,
3,6,3,2,4,
24,20,16] if Explicit then
spcreatetree(S)
spsetprobcond(2,prob)
forall(p in Products,s in 1..S) spsetrandatnode(yield(p),s,Values(p,s))
spgentree
else
spsetdist(union(p in Products) {yield(p)},Values,prob)
spgenexhtree
end-if
!
Model
TotalCost:= sum(p in Products) PlantingCost(p)*x(p) -
sum(p in AllProducts) SellingPrice(p)*w(p) +
sum(p in WheatAndCorn) PurchasePrice(p)*y(p)
LandUtilized:= sum(p in Products) x(p) <= TotalAvailableLand
forall(p in WheatAndCorn) MinCattleFeedRequirement(p):= yield(p)*x(p)+y(p)
w(p) >= MinimumRequirement(p)

SugarBeetProduction:= sum(p in {"SugBtFav","SugBtUnfav"})

w(p) <= yield("SugBt")*x("SugBt")

SugarBeetQuota:= w("SugBtFav") <= MaxSugarBeetQuota

forall(p in Products) spsetstage(x(p),"Wint")
forall(p in WheatAndCorn) spsetstage(y(p),"Summ")
forall(p in AllProducts) spsetstage(w(p),"Summ")

! Analysis

declarations

Problem={"Rec","Exp Val","Exp Val wt rec","Perf Inf","Perf Inf wt rec"}
end-declarations
forall(problem in Problem) Analyse(problem)

!***************************************************************

procedure Analyse(problem:string)

write("Solving " + problem + " Problem")

case problem of

"Exp Val": minimize(TotalCost,XSP_EV)
"Exp Val wt rec": minimize(TotalCost,XSP_EV+XSP_REC)
"Perf Inf": minimize(TotalCost,XSP_PI)
"Perf Inf wt rec": minimize(TotalCost,XSP_PI+XSP_REC)
"Rec": minimize(TotalCost,XSP_REC)
end-case

writeln("------------------------------------------")

write("Culture |")
forall(p in Products) write(strfmt(p,7))
writeln("------------------------------------------")

if(problem="Perf Inf") then
forall(s in 1..spgetscencount) do
write("Surface (acres) ",s," | ");
forall(p in Products) write(strfmt(getsolinscen(x(p),s),7,2));
writeln
end-do
else
write("Surface (acres) | ");
forall(p in Products) write(strfmt(speval(x(p)),7,2))
writeln
end-if
writeln("TotalProfit=$",-getobjval)
writeln("------------------------------------------")

case problem of
"Exp Val wt rec","Perf Inf wt rec":
do
forall(p in Products) spunfix(x(p))
sphethidden(LandUtilized,false)
doinend-do
end-case
end-procedure
end-model

5.1.2 Results

The overall expected profit is $108,390. The solution, which was obtained considering all circumstances, is ideal. It also allows selling some sugar beet at an unfavorable price or having unused quota and hence hedging against various scenarios. The model also provides the option of either creating the scenario tree explicitly or using the joint distribution functionality in mmsp. The following output is obtained after the model is run in Xpress-SP.

Solving Rec Problem
------------------------------------------
Culture | Wht Crn SugBt
------------------------------------------
### Surface (acres)

<table>
<thead>
<tr>
<th>Surface (acres)</th>
<th>170.00</th>
<th>80.00</th>
<th>250.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalProfit</td>
<td>$1,083,900</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Solving Exp Val Problem

<table>
<thead>
<tr>
<th>Culture</th>
<th>Wht</th>
<th>Crn</th>
<th>SugBt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface (acres)</td>
<td>120.00</td>
<td>80.00</td>
<td>300.00</td>
</tr>
<tr>
<td>TotalProfit</td>
<td>$1,186,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**mmsp warning 101:** fixing variables and hiding constraints

**mmsp warning 102:** hiding constraint, since all variables are hidden/fixed

#### Solving Exp Val wt rec Problem

<table>
<thead>
<tr>
<th>Culture</th>
<th>Wht</th>
<th>Crn</th>
<th>SugBt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface (acres)</td>
<td>120.00</td>
<td>80.00</td>
<td>300.00</td>
</tr>
<tr>
<td>TotalProfit</td>
<td>$1,072,400</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Solving Perf Inf Problem

<table>
<thead>
<tr>
<th>Culture</th>
<th>Wht</th>
<th>Crn</th>
<th>SugBt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface (acres) 1</td>
<td>183.33</td>
<td>66.67</td>
<td>250.00</td>
</tr>
<tr>
<td>Surface (acres) 2</td>
<td>120.00</td>
<td>80.00</td>
<td>300.00</td>
</tr>
<tr>
<td>Surface (acres) 3</td>
<td>100.00</td>
<td>25.00</td>
<td>375.00</td>
</tr>
<tr>
<td>TotalProfit</td>
<td>$1,154,006</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**mmsp warning 101:** fixing variables and hiding constraints

**mmsp warning 102:** hiding constraint, since all variables are hidden/fixed

#### Solving Perf Inf wt rec Problem

<table>
<thead>
<tr>
<th>Culture</th>
<th>Wht</th>
<th>Crn</th>
<th>SugBt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface (acres)</td>
<td>134.44</td>
<td>57.22</td>
<td>308.33</td>
</tr>
<tr>
<td>TotalProfit</td>
<td>$1,037,177</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

### 5.2 Finance

Stochastic programming has been extensively studied and applied in various sectors of finance, e.g., asset liability management, portfolio management, financial planning, etc. Here we consider a simple stochastic problem for financial planning [BL97b].

#### 5.2.1 Model description

We currently want to invest $b in stocks and bonds. At the end of T-1 years, we would like to exceed the goal $G$. The investment is changed every year. The excess income would imply an income of q% of the excess income and shortage would imply borrowing at r% of the amount short. The plot of the utility function for wealth at the end would look as follows:

We also assume that at each stage the return on stocks and bonds is 1.25 and 1.14 or 1.06 and 1.12 each with probability 0.5

- Decision variables
- \( x_t \): amount invested in asset \( i \) at time \( t \)
- \( y \): shortage
- \( w \): excess

- Constraints
  - Balance of initial investment, returns and goal meeting requirements

### 5.2.2 Xpress model

```plaintext
model stochasticStockAndBondModel
uses "mmsp"
parameters
T=4
Explicit=true
end-parameters

forward procedure generateTree

declarations
Assets="Stocks","Bonds"
Stages=1..T
x: array(Assets,1..T-1) of spvar ! Amount invested
w,y: spvar ! Shortage, excess
return: array(Assets,2..T) of sprand ! Return on investment
b,G: real ! Initial wealth, goal(in $1000)
q,r: ! Excess,shortage(in %)
ExpectedUtility: splinctr
Balance: array(1..T) of splinctr
end-declarations

! Data
b:=55;G:=80;q:=1;r:=4
spsetstages(Stages)
forall(i in Assets,t in 2..T) spsetstage(return(i,t),t)
generateTree
forall(i in Assets,t in 1..T-1) spsetstage(x(i,t),t)
spsetstage(y,T)
spsetstage(w,T)
ExpectedUtility:=q*y-r*w
forall(t in 1..T)
  if t=1 then
    Balance(t):=sum(i in Assets) x(i,t)=b;
```

---

Illustrative examples 43 Xpress-SP Reference Manual
elif t=T then
  Balance(t):=sum(i in Assets) return(i,t)*x(i,t-1) - y+w=G;
else Balance(t):=-sum(i in Assets) x(i,t)+
  sum(i in Assets) return(i,t)*x(i,t-1)=0;
end-if

setparam("xsp_scen_based",false)
setparam('xsp_verbose',false)
maximize(ExpectedUtility)

ObjvalRP:=getobjval
writeln("Obj value in recourse problem = ", strfmt(ObjvalRP,0,2))
forall(t in 1..T-1) spfix(x("Bonds",t),0)
maximize(ExpectedUtility)
writeln("Obj value in recourse problem = ", strfmt(getobjval,0,2))
writeln("VSS (if all investments in Stocks only) = ", strfmt(ObjvalRP-
  getobjval,0,2))
forall(t in 1..T-1) spunfix(x("Stocks",t))

!***********************************************************
procedure generateTree
declarations
  SetSrnd: set of sprand
  val: array(1..2,1..2) of real
  prob: array(1..2) of real
  StockVal,BondVal,BranchProb: array(1..2) of real
end-declarations
if(Explicit) then
  spcreatetree(2) ! Binary tree
  StockVal:=[1.25,1.06]
  BondVal:=[1.14,1.12]
  BranchProb:=[.5,.5]
  forall(t in 2..T) do ! Set Srnds
    spsetrand(return("Stocks",t),StockVal)
    spsetrand(return("Bonds",t),BondVal)
  end-do
  spsetprobcond(BranchProb) ! Set branch probabilities
  spgentree ! Generate tree
else ! Use joint distribution functionality
  val:=[1.25, 1.06, 1.14, 1.12]
  prob:=[.5, .5]
  forall(t in 2..T) do
    SetSrnd:={} ! Set Srnds
    forall(i in Assets) SetSrnd+={return(i,t)}
    spsetdist(SetSrnd,val,prob)
  end-do
  spgenexhtree
end-if
end-procedure
end-model

Note that in the above model, users may implicitly set the stages of return and x by setting
the control parameter xsp_implicit_stage to true, but this would also set the variables
w and y to the first stage, therefore their stages have to be set explicitly using the function
spsetsatage().

5.2.3 Results

The optimal value of expected utility is –1.514. The initial investment in stocks and bonds
is $41479 and $13520 respectively. The model also demonstrates how users may fix certain
decisions and solve the problem in the stochastic framework. In the above model, the impact
of placing all the funds in stocks is evaluated. It is observed that the expected utility using this
strategy is -3.79. Other results can be analyzed in IVE.
Figure 5.2: Stocks and bond model in IVE
Stochastic programming also finds wide-spread application in the energy sector where the demand for power and its prices are constantly changing. It has been used for solving various problems such as capacity expansion, unit commitment, etc. Here, we discuss an implementation of a capacity expansion model using Xpress-SP.

5.3.1 Model description

In a power-generating plant, the trend of yearly power demand looks as follows:

Here, for simplicity, we consider three modes of operations; namely low, average, and high. Clearly, the trend of power demanded in each year may vary depending on various factors such as rainfall, weather conditions, etc. We also consider three technologies (old, current, and new). Capacity expansion plans and the operational decisions need to be laid out for next three years. The objective is to minimize the expected total investment and production cost. The stochastic model entities are described below.

- **Parameters**
  - Stages = 1,...,4 : indexed by $t$
  - Technology = {old,curr,new} : indexed by $i$
  - Modes=low,avg,high : indexed by $j$

- **Data**
  - $g_i$: existing capacity of technology $i$ added at time $t$
  - $r_{it}$: investment cost per unit of capacity of technology $i$ added at time $t$
  - $q_i$: production cost per unit of capacity of technology $i$ added at time $t$

- **Decision variables**
  - $x_{it}$: capacity of technology $i$ added at time $t$
  - $y_{ijt}$: capacity of type $i$ dedicated in mode $j$ at time $t$

- **Constraints**
  - Accumulated Capacity balance

Illustrative examples 46 Xpress-SP Reference Manual
Figure 5.4: Production planning schedule

- Demand balance

- Objective
  - minimize the expected total investment and production cost

The capacities are added in the first two years and hence the total available capacities are realized in the second and the third year. In the years two through four, the available capacities are distributed among different modes.

The scenarios are created by discretizing the load-duration curve. For simplicity, the duration in each mode is considered one-third of a year. We present three ways of generating scenarios:

1. Exhaustive: The demand (low, average, and high) has an independent distribution in each year.
2. Stage symmetric: There are 8, 5, and 3 load-duration curves possible in the first, second, and third year respectively.
3. Explicit: The demand in the subsequent year depends on the earlier years’ demand and hence, all possible realizations are explicitly defined.

5.3.2 Exhaustive model

```plaintext
model CapacityExpansion
uses 'mmsp'
parameters
T=4           ! Years
end-parameters
```
forward procedure generateExhaustiveTree

declarations
Stages=1..T
Technology="old","curr","new"
Modes="low","avg","high"

r: array(Technology,1..T-2) of real ! Investment cost in $1000/MW cap. added
g: array(Technology,1..T-1) of real ! Existing capacity in MW
q: array(Technology,2..T) of real ! Unit production cost in $1000/MWh
tau: real ! Time in hour spent in each mode
Budget: real ! Max. additional capacity in MW
Penalty: real ! For not meeting demand in $1000/MW
d: array(Modes,2..T) of sprand ! Total demand in MW
umd: array(Modes,1..T-1) of spvar ! Total unmet demand in MW
w: array(Technology,2..T-1) of spvar ! Total available capacity in MW
y: array(Technology,Modes,2..T) of spvar ! Capacity used in production in MW
x: array(Technology,1..T-2) of spvar ! Additional capacity in MW
Cost: splinctr
BudgetSpent: splinctr
AccumulatedCapacity: array(Technology,2..T-1) of splinctr
Demand: array(Modes,1..T-1) of splinctr
Capacity: dynamic array(Technology,1..T-1) of splinctr
end-declarations

! Read in data from external file
initializations from 'CapacityExpansion.dat'
r q g tau Budget Penalty
end-initializations

setparam('xsp_implicit_stage',true)
spsetstages(Stages)
generateExhaustiveTree

Cost:= sum(t in 1..T-2,i in Technology) r(i,t)*x(i,t) +
sum(t in 2..T,i in Technology,j in Modes) q(i,t)*tau*y(i,j,t) +
sum(j in Modes,t in 1..T-1) Penalty*umd(j,t)
BudgetSpent:= sum(i in Technology,t in 1..T-2) x(i,t)<=Budget
forall(i in Technology,t in 2..T-1) if (t-1>=2) then
  AccumulatedCapacity(i,t):= w(i,t)=x(i,t-1)+w(i,t-1);
else
  AccumulatedCapacity(i,t):= w(i,t)=x(i,t-1)
end-if
forall(j in Modes,t in 1..T-1) Demand(j,t):= umd(j,t) + sum(i in Technology) y(i,j,t+1)>=d(j,t+1)
forall(i in Technology,t in 1..T-1) if(t>=2) then
  Capacity(i,t):=sum(j in Modes) y(i,j,t+1)<=g(i,t)+w(i,t));
else
  Capacity(i,t):=sum(j in Modes) y(i,j,t+1)<=g(i,t)
end-if
forall(t in 2..T) OperatingCost(t):=sum(i in Technology, j in Modes)
q(i,t)*tau*y(i,j,t)
minimize(Cost)
writeln(getobjval)

!*******************************************************************************
procedure generateExhaustiveTree

declarations
E=3 ! Max no. of values, each demand can assume
val,prob:array(Modes,2..T,1..E) of real
end-declarations

initializations from 'CapacityExpansionExhaustive.dat'
val prob
end-initializations

declarations
vals,probs:array(1..E) of real
Illustrative examples 48 Xpress-SP Reference Manual
forall(j in Modes, t in 2..T) do 
forall(e in 1..E) do 
vals[e] := val(j, t, e) 
probs[e] := prob(j, t, e) 
end-do 
spsetdist(d(j, t), vals, probs) 
end-do 
spgenexhtree 
end-procedure 
end-model 

Data files: 

CapacityExpansion.dat:

r: [464, 500, 508, 507, 534, 490] 
q: [.040, .034, .032, .030, .022, .021, .019, .018, .016] 
g: [80, 50, 20, 70, 35, 20, 40, 25, 15] 
tau: 2900 
Budget: 350 
Penalty: 1300

CapacityExpansionExhaustive.dat:

val: [20, 25, 0, 28, 33, 0, 36, 40, 0, 50, 55, 0, 58, 63, 0, 66, 70, 0, 80, 85, 0, 88, 93, 0, 96, 100, 0] 
! replace zeros by some other numbers to get more possible realizations and scenarios. 
prob: [.5, .5, 0, .5, .5, 0, .4, .6, 0, .6, .4, 0, .4, .6, 0, .6, .4, 0, .4, .6, 0, .5, .5, 0, .5, .5, 0] 

5.3.3 Stage symmetric model 

In order to generate a scenario tree, which is symmetric with respect to stage, the procedure generateExhaustiveTree in the above example can be replaced by the procedure generateStageSymmetricTree which is defined below. The parameter Dep when set to true, would create the tree using the joint probability distribution facility in mmsp.
NumBranch: array(2..T) of integer
end-declarations

initializations from DIR+"CapacityExpansionStage.dat"
NumBranch
end-initializations

MaxBranch:=max(t in 2..T) NumBranch(t)
declarations
BranchProb: array(2..T, 1..MaxBranch) of real
BranchValues: array(Modes, 2..T, 1..MaxBranch) of real
Prob, Values: array(1..MaxBranch) of real
end-declarations

initializations from DIR+"CapacityExpansionStage.dat"
BranchProb BranchValues
end-initializations
declarations
AllDem: set of sprand
Val: array(Modes, 1..MaxBranch) of real
end-declarations

if(Dep) then
forall(t in 2..T) do
AllDem:=union(j in Modes) {d(j,t)}
forall(b in 1..MaxBranch) Prob(b):=0
forall(b in 1..NumBranch(t)) Prob(b):=BranchProb(t,b)
forall(j in Modes, b in 1..MaxBranch) Val(j,b):=0
forall(j in Modes, b in 1..NumBranch(t)) Val(j,b):=BranchValues(j,t,b)
spsetdist(AllDem, Val, Prob)
end-do
spgenexhtree
else
spcreatetree(NumBranch)
forall(t in 2..T) do
forall(b in 1..NumBranch(t)) Prob(b):=BranchProb(t,b)
spsetprobcond(t, Prob)
end-do
forall(t in 2..T, j in Modes) do
forall(b in 1..NumBranch(t)) Values(b):=BranchValues(j,t,b)
spsetrand(d(j,t), Values)
end-do
spgentree
end-if
end-procedure

CapacityExpansionStage.dat:

NumBranch: [8, 5, 3]
BranchProb: [0.15, 0.15, 0.1, 0.1, 0.2, 0.1, 0.1, 0.1,
0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0,
0.25, 0.25, 0, 0, 0, 0]
BranchValues: [80, 81, 82, 83, 84, 85, 86, 87, 88,
86, 88, 90, 93, 95, 0, 0, 0,
96, 98, 100, 0, 0, 0, 0, 0,
50, 51, 52, 53, 54, 56, 57, 58,
56, 58, 60, 63, 65, 0, 0, 0,
66, 68, 70, 0, 0, 0, 0, 0,
20, 21, 22, 23, 24, 25, 26, 27, 28,
26, 28, 30, 33, 35, 0, 0, 0,
36, 38, 40, 0, 0, 0, 0, 0]

5.3.4 Explicit model

A scenario tree with seven scenarios in the above example can be generated as follows:

procedure generateExplicitTree
declarations
S=7 ! No. of scenarios
A: array(1..S,1..T) of integer ! Node numbers in tree
N: array(1..T) of integer ! Number of nodes in each stage
end-declarations

initializations from DIR+'CapacityExpansionExplicit.dat'
A N end-initializations
spcreatetree(A)
Nmax:=max(t in 1..T) N(t)
declarations
val: array(2..T,Modes,1..Nmax) of real
prob: array(2..T,1..Nmax) of real
end-declarations
initializations from DIR+'CapacityExpansionExplicit.dat'
val prob end-initializations
forall(t in 2..T,n in 1..N(t)) spsetprobcondatnode(t,n,prob(t,n))
forall(t in 2..T,j in Modes,n in 1..N(t))
spsetrandatnode(d(j,t),n,val(t,j,n))
spgentree
end-procedure

CapacityExpansionExplicit.dat:

A: [1,1,1,1,
  1,1,2,2,
  1,1,2,3,
  1,1,2,4,
  1,2,3,5,
  1,2,3,6,
  1,3,4,7]
N: [1,3,4,7]
val: [20,25,30,0,0,0,0,
  50,55,60,0,0,0,0,
  30,60,90,0,0,0,0,
  25,35,30,0,0,0,0,
  55,65,60,60,0,0,0,
  85,95,90,90,0,0,0,
  38,35,38,40,36,39,38,
  68,65,68,70,66,68,68,
  98,95,98,100,96,99,98] prob: [.333,.333,.333,0,0,0,0,
  .5,.5,1,1,0,0,0,
  1,.333,.333,.333,.5,.5,1]

5.3.5 Results

The results for each of these models with respective scenario generation schemes can be obtained using the standard functions available in the mmsp module and IVE. The plot of other entities, e.g., the operating cost in stage 3, can be viewed in IVE (see Figure 5.5).

5.4 Supply chain management

Consider a supply chain problem in which a company makes products: shirts and trousers and then ships them to different destinations: New York and Los Angeles. The company needs to make production decisions well in advance, and when the demand of each product at each destination occurs, the company needs to make decisions for the next three periods.
Figure 5.5: Operating cost in third stage (scenario generation scheme: exhaustive)

Figure 5.6: Products and destinations layout
5.4.1 Model description

- Decisions
  - Before demand occurs: How much of each of the products to be bought
  - After demand occurs: How much of each of the products to be supplied to each destination

- Constraints
  - Available man-hours
  - Demand balance constraints

- Objective: Maximize net profit = Expected (Revenue on sales – cost of lost sales – cost of excess supply – cost of procurement)

- Variable $x_{it}$: Quantity of product $i$ made in the plant at stage $t$ (beginning of $t^{th}$ period)
- Parameter $B$: Available man-hours at the plant
- Parameter $a_i$: time required on product $i$
- Variable $y_{ijt}$: Amount of product $i$ supplied to destination $j$ at stage $t$
- Variables $l_{sijt}$, $e_{sijt}$: lost sales and excess supply of product $i$ at destination $j$ at stage $t$.

Demand being a random variable may have a distribution, e.g., as shown below.

The company also wants to take into consideration such randomness and hedge against the uncertain future by making an overall optimal decision. The stochastic model looks as follows:
5.4.2 Xpress model

model StochasticMultiStageSupplyChain
uses 'mmsp','mmdbdc'
parameters
T=3
DIR=''
end-parameters

declarations
CSTR='DSN=MS Access Database; DBQ='+DIR+'SupplyChain.mdb'
Products,Destinations:set of string
S,CLS,CES: dynamic array(Products,Destinations) of real
C,a: dynamic array(Products) of real
B: real
end-declarations

!--------------------------sets Data Values ----------------------
setparam("SQLverbose",true)
SQLconnect(CSTR)
SQLexecute("select Products,Destinations,SellingPrice, CostOfLostSales, CostOfExcessSales from DETERMINISTIC_DATA_2INDEX",[S,CLS,CES])
finalize(Products);finalize(Destinations)
SQLexecute("select Products,CostPrice, TimePerUnit from DETERMINISTIC_DATA_1INDEX",[C,a])
B:=SQLreadreal("select AvailableTime from DETERMINISTIC_DATA_0INDEX")

!--------------------------Stochastic Data values-------------------
declarations
n: array(Products,Destinations) of integer ! number of discretized ! points for distr. of demand
end-declarations
SQLexecute("select Products,Destinations,NoOfDiscretizedPoints from EXHAUSTIVE_GENERATION_PARAMS", n)
n_max:=max(i in Products,j in Destinations) n(i,j)
declarations
DiscretizedValueDemand,DiscretizedProbabilityDemand:
array(Products,Destinations,1..n_max) of real
end-declarations
SQLexecute("select Products,Destinations,DiscretizedPoint,ValueDemand, ProbabilityDemand from RANDELEMDISTN", [DiscretizedValueDemand, DiscretizedProbabilityDemand])
SQLdisconnect

!---------------------------- Stochastic Model declarations -----------
setparam('xsp_implicit_stage',true)
declarations
Stages=1..T
x: array(Products,1..T-1) of spvar
y,ls,es: array(Products,Destinations,2..T) of spvar
D: array(Products,Destinations,2..T) of sprand
TotalProfit: spinctr
Time: array(1..T-1) of spinctr
Balance: array(Products,2..T) of spinctr
Demand: array(Products,Destinations,2..T) of spinctr
end-declarations

!---------------------------- Random variables ------------------------
spsetstages(Stages)
declarations
val,prob:array(1..n_max) of real
end-declarations
forall(i in Products,j in Destinations,t in 2..T) do
forall(k in 1..n(i,j)) do
val(k):=DiscretizedValueDemand(i,j,k)
prob(k):=DiscretizedProbabilityDemand(i,j,k)
end-do
spsetdist(D(i,j,t),val,prob)
end-do
spgenexhtree

!---------------------------- Model --------------------------

TotalProfit:= -sum(i in Products,t in 1..T-1) C(i)*x(i,t)+
sum(i in Products,j in Destinations,t in 2..T)
(S(i,j)*y(i,j,t)-CLS(i,j)*ls(i,j,t)-CES(i,j)*es(i,j,t))

forall(t in 1..T-1) Time(t):=sum(i in Products) a(i)*x(i,t)<=B
forall(i in Products,t in 2..T) Balance(i,t):=
sum(j in Destinations) y(i,j,t)=x(i,t-1)
forall(i in Products,j in Destinations,t in 2..T)
if(t>2) then
  Demand(i,j,t):=y(i,j,t)+ls(i,j,t)-es(i,j,t)+es(i,j,t-1)=D(i,j,t)
else
  Demand(i,j,t):=y(i,j,t)+ls(i,j,t)-es(i,j,t)=D(i,j,t)
end-if

maximize(TotalProfit)
exit(0)
end-model

5.4.3 Data

- DETERMINISTIC_DATA_0INDEX
  AvailableTime
    165

- DETERMINISTIC_DATA_1INDEX
  Products  CostPrice  TimePerUnit
  Shirts    10         1
  Trousers  20         2

- DETERMINISTIC_DATA_2INDEX
  Products  Destinations  SellingPrice  CostOfLostSales  CostOfExcessSales
  Shirts    NYC            25          0.75            10
  Shirts    LA             20          0.25            8
  Trousers  NYC            45          1.5             15
  Trousers  LA             40          0.5             15

- EXHAUSTIVE_GENERATION_PARAMS
  Products  Destinations  NoOfDiscretizedPoints
  Shirts    NYC            2
  Shirts    LA             2
  Trousers  NYC            2
  Trousers  LA             2

- RANDELEMDISTN (consider 2 possibilities for each demand)
  Products  Destinations  DiscretizedPoint  ValueDemand  ProbabilityDemand
  Shirts    NYC            1             10             0.5
  Shirts    NYC            2             30             0.5
  Shirts    LA             1             25             0.5
  Shirts    LA             2             40             0.5
  Trousers  NYC            1             30             0.5
  Trousers  NYC            2             40             0.5
  Trousers  LA             1             20             0.5
  Trousers  LA             2             35             0.5

5.4.4 Results

The problem consists of 4096 scenarios. The expected profit is $5,557,731.934. The problem statistics, and the values and distribution of various entities of the model can be viewed in IVE. The figure below shows the distribution of 'Total Profit' across the scenarios.
5.5 Option pricing in presence of transaction costs

This model is based on the option pricing model as described in [Cor]. The author assumes a binomial approximation to the Black-Scholes Option Pricing Model to explain the effect of transaction costs on option prices. The behavior of stocks and bonds prices is considered over successive stages. Bonds have fixed annualized rate of return, while stock's prices go up/down depending on the market.

5.5.1 Model description

- Parameters:
  - \( T \) = Number of stages (indexed by \( t \))
  - Assets= (Stocks, Bonds) (indexed by \( i \))
  - \( r \) = Fixed annualized rate of return of bonds
  - \( \sigma \) = Volatility of returns of stocks
  - \( \Delta \) = Time span for re-investment
  - \( \theta \) = Transaction cost
  - \( X \) = Strike price of option

- Stochasticity:
  - \( \tilde{\delta}_t \) = Movement (1/-1 corresponding to up/down) of stock at stage \( t \)
  - \( \tilde{P}_{it} \) = Price of Asset \( i \) at stage \( t \):
    - \( P_{Bonds,t} = P_{Bonds,t-1} \cdot e^{r \Delta} \), \( P_{Stocks,t} = P_{Stocks,t-1} \cdot e^{\tilde{\delta}_t \sigma \sqrt{\Delta}} \)
  - \( \tilde{O} \) = Option payoff on maturity:
    - for call option \( \tilde{O} = (\tilde{P}_{Stocks,T} - X)^+ \), for put option \( \tilde{O} = (X - \tilde{P}_{Stocks,T})^+ \)

- Decision variables:
  - \( q_{it} \) = Quantity of Asset \( i \) at stage \( t \)
- $x_t, y_t$ = Quantity of stocks bought and sold respectively at stage $t$

- **Model formulation:**

  - $\min q, x, y \sum_{i \in \text{Assets}} q_{i,1} \cdot P_{i,1}$
  - s.t.
    - $q_{\text{Stock},t} - q_{\text{Stock},t-1} = x_t - y_t \forall t \in 2..T$
    - $(x_t(1 + \theta) - y_t(1 - \theta)) \cdot P_{\text{Stock},t} + (q_{\text{Bonds},t} - q_{\text{Bonds},t-1}) \cdot P_{\text{Bonds},t} \leq 0 \forall t \in 2..T - 1$
    - $\sum_{i \in \text{Assets}} q_{i,T-1} \cdot P_{i,T} \geq O$
    - $x_t, y_t \geq 0 \forall t \in 2..T$

### 5.5.2 Xpress model

```plaintext
model OptionPricing
uses 'mmsp'

parameters
T=5  ! No. of stages
call=true ! Call/ put option
X=100.0  ! Strike price
theta=0.02 ! Fraction of cost of trading
sigma=0.35 ! Volatility
delta=1.0 ! Time-span for re-investment
r=0.07  ! Fixed rate of return for bonds
InitStkPrice=95  ! Stock’s price initially
InitBndPrice=100  ! Bond’s price initially
end-parameters

forward procedure generate_tree  ! Generate tree
declarations
Assets = {"Stocks","Bonds"}  ! Set of all the assets
Stages = 1..T ! set of stages
q: array(Assets,Stages) of spvar ! Quantity of assets in each stage
x,y: array(2..T) of spvar  ! Quant of stocks bought, sold resp
movement: array(2..T) of sprand ! Up/down (1/-1 resp)
price: array(Assets,Stages) of sprandexp ! Price of assets
InitInv: splincr ! Total initial investment
BalanceStocks: array(2..T) of splincr ! Balance of stocks
BalancePosition: array(2..T) of splincr ! Balance of cash position
end-declarations

spsetstages(Stages) ! Set stages for SP
setparam('xsp_implicit_stage',true) ! Use last index to associate with stage
forall(t in Stages) ! Random price of assets in each stage
if t=1 then
  price("Stocks",t):=InitStkPrice
  price("Bonds",t):=InitBndPrice
else
  price("Stocks",t):=price("Stocks",t-1)*exp(movement(t)*sigma*delta^0.5)
  price("Bonds",t):=price("Bonds",t-1)*exp(r*delta)
end-if

if call then ! payoff at end depending on option type
  OptPayOff:=maximum(price("Stocks",T)-X,0)
else
  OptPayOff:=maximum(X-price("Stocks",T),0)
end-if

generate_tree  ! Generate stoch. tree based on movement
forall(i in Assets,t in Stages)
  q(i,t) is_free  ! Negative value implies short position

InitInv:=sum(i in Assets) q(i,1)*price(i,1)
forall(t in 2..T)
  BalanceStocks(t):= q("Stocks",t)-q("Stocks",t-1) = x(t)-y(t)
```

Illustrative examples 57 Xpress-SP Reference Manual
forall(t in 2..T)
if t=T then
    BalancePosition(t):= sum(i in Assets) q(i,t-1)*price(i,t) >= OptPayOff
else
    BalancePosition(t):=
        (x(t)+(1+theta)-y(t)+(1-theta))*price("Stocks",t) +
        (q("Bonds",t)-q("Bonds",t-1))*price("Bonds",t) <= 0
end-if

setparam("xsp_scen_based",false) ! Create a node based problem
minimize(InitInv)

********************************************************************************
procedure generate_tree
declarations
    val:array(1..2) of real
    prob:array(1..2) of real
end-declarations
val:=[1,-1] ! Up/down movement values
prob:=[.5,.5] ! Up/down movement probabilities
forall(t in 2..T) spsetdist(movement(t),val,prob) ! Set distribution
spgenexhtree ! Generate exh. tree based on distrib.s
end-procedure
end-model

5.5.3 Results

The model is solved by varying the value of θ from 0 to 0.1.

From the above figure it can be observed that as the transaction cost is increased, the initial investment also increases almost linearly. Additionally, with the increase in transaction cost, the initial quantity of stocks bought keeps increasing, while the quantity of bonds that is kept on a short position keeps decreasing. Other results may be analyzed using IVE.
Airlift operations scheduling model

This model is based on the airlift operation scheduling problem described by Ariyawansa and Felt in their collection of stochastic linear programming test problems [AF05]. The problem consists of scheduling airlift operations along a set of routes during a period of one month. The demands of routes can be predicted but are subject to changes. A random variable is assigned for each demand. If the predicted demands do not agree with the actual demands, a recourse action is taken.

To meet the route demands, several types of aircrafts are available. Each aircraft has its own characteristics, namely availability of number of flight hours per month and carrying capacity when flying a specific route.

The recourse actions that can be taken are to switch aircrafts from one route to another, to leave some flights unused, and to contract commercial flights to meet the demands of routes.

Let \( a_{ij} \) be the number of hours required by an aircraft of type \( i \) to complete a flight of route \( j \) and let \( x_{ij} \) be the number of flights planned for route \( j \) using aircraft of type \( i \) and the maximum number of flight hours for aircraft of type \( i \) during a month. Then, the constraint on the number of flight hours used by an aircraft of type \( i \) during the month is

\[
\sum_{i} a_{ij} \cdot x_{ij} \leq F_i
\]

Let \( a_{ijk} \) be the number of hours required by an aircraft of type \( i \) to fly route \( k \) after having been switched from route \( j \) and let \( x_{ijk} \) be the increase of flights in route \( k \) after switching an aircraft \( i \) from route \( j \). Then we have the following constraint that states that the number of flight hours switched from an aircraft of type \( i \) and from route \( j \) is at most the amount we have originally scheduled:

\[
\sum_{i} a_{ijk} \cdot x_{ijk} \leq a_{ij} \cdot x_{ij} \forall i, j
\]

Let the carrying capacity of aircraft of type \( i \) when flying in route \( j \) be denoted by \( b_{ij} \), the demand of route \( j \) commercially contracted in the recourse by \( y_j^+ \), and the unused capacity assigned to route \( j \) by \( y_j^- \). We can express a constraint that ensures that the flight hours initially scheduled for route \( j \), the flights switched from and to route \( j \), the commercially contracted demand, and the unused capacity sum up to the demand \( d_j \) of route \( j \) as follows:

\[
\sum_{i} b_{ij} \cdot x_{ij} - \sum_{k \neq j} b_{ij} \cdot \frac{x_{ik}}{a_{ii}} + \sum_{k \neq j} b_{ij} \cdot x_{ikj} + y_j^+ - y_j^- = d_j \quad \forall j
\]

Let \( c_{ij} \) be the cost of initially assigning an aircraft of type \( i \) to fly one flight on route \( j \), and let \( c_{ijk} \) be the cost for aircraft of type \( i \) to fly one flight of route \( k \) after having been switched from route \( j \). With \( c_j^+ \) denoting the cost of transporting one unit of demand of route \( j \) by a commercially contracted flight and denoting the cost of one unit of unused capacity on route \( j \), we can define the model as follows:
The following table describes the constraints of the model, the data, and the decision variables (with its corresponding names in the Mosel code).

Table 5.1: Model identifiers

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_{ij}) ((a))</td>
<td>number of hours required by aircraft of type (i) to complete a flight of route (j)</td>
</tr>
<tr>
<td>(a_{ijk}) ((Aswitch))</td>
<td>as the number of hours required by aircraft of type (i) to fly route (k) after having been switched from route (j)</td>
</tr>
<tr>
<td>(x_{ij}) ((x))</td>
<td>number of flights planned for route (j) using aircraft of type (i)</td>
</tr>
<tr>
<td>(x_{ijk}) ((xswitch))</td>
<td>increase of flights in route (k) after switching aircraft (i) from route (j)</td>
</tr>
<tr>
<td>(F_{i}) ((F))</td>
<td>maximum number of flight hours for aircraft of type (i) during a month</td>
</tr>
<tr>
<td>(c_{ij}) ((Cost))</td>
<td>cost of initially assigning aircraft of type (i) to fly one flight of route (j)</td>
</tr>
<tr>
<td>(c_{ijk}) ((CostSwitch))</td>
<td>cost for aircraft of type (i) to fly one flight of route (k) after having been switched from route (j)</td>
</tr>
<tr>
<td>(c_{i}^+) ((cplus))</td>
<td>cost of transporting one unit of demand of route (j) by a commercially contracted flight</td>
</tr>
<tr>
<td>(c_{i}^-) ((cminus))</td>
<td>cost of one unit of unused capacity on route (j)</td>
</tr>
<tr>
<td>(y_{ij}^+) ((yplus))</td>
<td>demand of route (j) commercially contracted in the recourse</td>
</tr>
<tr>
<td>(y_{ij}^-) ((yminus))</td>
<td>unused capacity assigned to route (j)</td>
</tr>
<tr>
<td>(b_{ij}) ((b))</td>
<td>carrying capacity of aircraft of type (i) when flying in route (j)</td>
</tr>
<tr>
<td>(d_{j}) ((d))</td>
<td>demand of route (j)</td>
</tr>
<tr>
<td>MaxHours ((i))</td>
<td>constraints (1) of the model</td>
</tr>
<tr>
<td>MaxSwitch ((i,j))</td>
<td>constraints (2) of the model</td>
</tr>
<tr>
<td>DemandCtr ((j))</td>
<td>constraints (3) of the model</td>
</tr>
<tr>
<td>ExpTotalCost</td>
<td>objective function</td>
</tr>
</tbody>
</table>

5.6.1 Implementation

Below, we show the Mosel code that implements the model and solves it using Xpress-SP.

```mosel
model airlift
```

Illustrative examples 60 Xpress-SP Reference Manual
uses "mssp"

parameters
  DIR=":"
end-parameters

declarations
  NAIRCRAFTS = 2
  NROUTES = 2
  SCEN = 25
  AIRCRAFTS = 1..AIRCRAFTS
  ROUTES = 1..NROUTES
  A : array(AIRCRAFTS,ROUTES) of integer
  Aswitch : array(AIRCRAFTS,ROUTES,ROUTES) of integer
  F : array (AIRCRAFTS) of integer
  b : array (AIRCRAFTS,ROUTES) of integer
  Cost : array (AIRCRAFTS,ROUTES) of real
  CostSwitch: array (AIRCRAFTS,ROUTES,ROUTES) of real
  cplus : array (ROUTES) of real
  cminus : array (ROUTES) of real
  Probabilities : array (1..SCEN) of real
  DValues : array (ROUTES,1..SCEN) of real
  d : array (ROUTES) of sprand
  MaxHours : array (AIRCRAFTS) of splinctor
  MaxSwitch : array (AIRCRAFTS, ROUTES) of splinctor
  DemandCtr : array (ROUTES) of splinctor
  x : array (AIRCRAFTS,ROUTES) of spvar
  xswitch : dynamic array (AIRCRAFTS,ROUTES,ROUTES) of spvar
  yplus : array (ROUTES) of spvar
  yminus : array (ROUTES) of spvar
end-declarations

! Read data
initializations from DIR+'airlift.dat'
  A Aswitch F b Cost CostSwitch
  cplus cminus DValues Probabilities
end-initializations

!---------------------------------------------------------
! Create scenario tree
declarations
  Stages = {"First","Recourse"}
  Branches: integer
  Values : array (1..SCEN) of real
end-declarations
spsetstages(Stages)
forall(i in ROUTES) spsetstage(d(i),"Recourse")
spcreatetree(SCEN)
speauprobcnd(2,Probabilities)
forall(i in ROUTES, j in 1..SCEN) spsetrandatnode(d(i),j,DValues(i,j))
spgenmtree

!---------------------------------------------------------
forall (i in AIRCRAFTS, j in ROUTES, k in ROUTES | k<>j)
create(xswitch(i,j,k))

ExpTotalCost :=
  sum(i in AIRCRAFTS, j in ROUTES) Cost(i,j)*x(i,j) +
  sum(i in AIRCRAFTS, j in ROUTES, k in ROUTES | k<>j)
  (CostSwitch(i,j,k) - Cost(i,j)*Aswitch(i,j,k)/A(i,j))*xswitch(i,j,k)+
  sum(j in ROUTES) cplus(j)*yplus(j) +
  sum(j in ROUTES) cminus(j)*yminus(j)

! First stage constraints:
forall (i in AIRCRAFTS)
  MaxHours(i):=
  sum(j in ROUTES) A(i,j)*x(i,j) <= F(i)

! Second stage constraints:
forall (i in AIRCRAFTS, j in ROUTES)
  MaxSwitch(i,j):=
Figure 5.9: Schedule

\[
\sum_{k \in \text{ROUTES} | k \neq j} A_{\text{switch}}(i, j, k) \cdot x_{\text{switch}}(i, j, k) \\
\leq A(i, j) \cdot x(i, j)
\]

! Demand constraint for recourse
forall \(j \in \text{ROUTES} \) 
\[
\text{DemandCtr}(j) := -\sum_{i \in \text{AIRCRAFTS}, k \in \text{ROUTES} | k \neq j} (b(i, j) \cdot A_{\text{switch}}(i, j, k) / A(i, j)) \cdot x_{\text{switch}}(i, j, k) + \\
\sum_{i \in \text{AIRCRAFTS}, k \in \text{ROUTES} | k \neq j} b(i, j) \cdot x_{\text{switch}}(i, k, j) + \\
y_{\text{plus}}(j) - y_{\text{minus}}(j) = \\
d(j) - \sum_{i \in \text{AIRCRAFTS}} b(i, j) \cdot x(i, j)
\]

! Associate svars to stages
forall(i \in \text{AIRCRAFTS}, j \in \text{ROUTES}) spsetstage(x(i, j), 'First')
forall(i \in \text{AIRCRAFTS}, j \in \text{ROUTES}, k \in \text{ROUTES} | k \neq j) 
spsetstage(x_{\text{switch}}(i, j, k), 'Recourse')
forall(j \in \text{ROUTES}) spsetstage(y_{\text{plus}}(j), 'Recourse')
forall(j \in \text{ROUTES}) spsetstage(y_{\text{minus}}(j), 'Recourse')

! Set variables as integer
forall(i \in \text{AIRCRAFTS}, j \in \text{ROUTES}) x(i, j) is_integer
forall(i \in \text{AIRCRAFTS}, j \in \text{ROUTES}, k \in \text{ROUTES} | exists(x_{\text{switch}}(i, j, k))) 
\quad x_{\text{switch}}(i, j, k) is_integer

setparam('xprs_cutfreq', 1)
setparam('xprs_cutdepth', 40)
setparam('xprs_treegomcuts', 1)
setparam('xprs_varselection', 3)
minimize(ExpTotalCost)
writeln(getobjval)
end-model

### 5.6.2 Results

Here, we present results for a small example, with two types of aircrafts, two routes and twenty-five scenarios. The expected total cost is $229,431. The values of the variables \(x_{ij}\) and \(x_{ijk}\) are shown in the figure below for a specific scenario (scenario 7). The solution is shown in the following picture, where A-B corresponds to route 1 and C-D to route 2. The pairs \((a, b)\) give the number of aircrafts of type 1 and 2, respectively, that are initially assigned to each route. The pairs \([a, b]\) give the number of aircrafts of type 1 and 2 respectively, that are switched from one route to another in the recourse stage, depending on the direction of the arrow.

The data file airlift.dat is shown in the Appendix. Figure 5.10 shows the model and its associated scenario tree in Xpress-IVE.
Figure 5.10: Airlift model in Xpress-IVE.
### 5.6.3 Analysis

An interesting way of measuring the quality of the stochastic solution of the model airlift is to compare its result to that of the expected value version of the model. To solve the expected value problem in Xpress-SP the model shown above can be used. The only difference is that when optimizing, the type of problem that must be solved is specified:

```plaintext
minimize(ExpTotalCost,XSP_EV+XSP_REC)
! Expected value with recourse
```

The expected value version of the model has an optimal objective function value of $237,016. Thus, the value of the stochastic solution is

$$VSS = Z_{EVr} - Z_R = 7585$$

Other useful techniques that can be easily applied in Xpress-SP consist of scenario aggregation and deletion. For instance, one may want to delete/aggregate scenarios that are unlikely to happen and some scenarios that describe similar realizations. In these situations, it may be interesting to delete scenarios or aggregate some of them, in order to simplify the problem. This task can be performed using Xpress-IVE using the Run options window of Xpress-IVE (Menu Build ➪ Options). The option ‘Pause to prune scenario tree manually’ allows the user to select some scenarios (as show in Figure 5.11), delete them, and resume execution, in which case the problem is solved with the current scenario tree. Aggregation can be done in a similar way.

As an example, suppose that the user decides to delete the scenarios related to the lowest demand predictions. If we delete the seven scenarios with lowest demands for the second route (by using the procedure `spdelscen((2,8,9,10,12,18,23))`, we end up with a problem with 18 scenarios, which gives expected total cost $233,389. If we do a similar exercise, but removing the 7 scenarios with highest demand prediction for the second route (`spdelete((1,3,7,11,13,14,20))`), we have an expected total cost equal to $222,258.

### 5.7 Forest planning

This model is based on the forest planning problem described by Ariyawansa and Felt in their collection of stochastic linear programming problems [AF05]. The goal is to maximize the value of timber, both cut and remaining, after the last period of planning. The constraints in this model pertain to the amount of forest cut in a single period, the age of trees, and the likelihood that trees remaining uncut are destroyed by fire.

Each tree belongs to one of $K$ age classes of equal length. The length that defines the age classes is also used to define the length of stages of the planning period. Thus, trees belonging to age class $A$ in a given stage will belong to age class $A + 1$ in the next stage if they are not cut or destroyed by fire. Whenever any of these events take place, new trees are planted and, the number of trees in the first age class equals the sum of these two quantities.

The variables of the model are the total area of forest for each age class, for each period (vector $s_t$), the amount of forest harvested for each age class in each period (vector $x_t$), and penalty factors for not satisfying constraints on the amount of harvested area in each period (vectors $p_{t1}$ and $p_{t2}$).

A natural constraint is one that restricts the harvested area to be at most the total amount of area for each age class in each period $t = 1, \ldots, T$:

$$x_t \leq s_t$$

The next set of constraints on the harvested area involve penalty factors $p_{t1}$ and $p_{t2}$ which try to avoid the harvested quantities exceeding limits of how fast the timber industry can change its purchasing volume from the current period to the next:
Figure 5.11: Pruning scenario tree in Xpress-IVE.
\[ \alpha y^T x_{t-1} - y^T x_t \leq p_{t1} \]
\[ y^T x_{t-1} - \beta y^T x_{t-1} \leq p_{t2} \]

In these constraints \( \alpha \) and \( \beta \) are constants and \( y \) is a vector representing the yield of selling one unit of harvested area of each class.

The matrices \( Q \) and \( P_t \) shown below are square matrices of dimension \( K \) that are used to express the total number of trees in each period for each age class, taking into account both harvesting and fire in the previous stage.

\[
Q = \begin{bmatrix}
1 & 1 & \cdots & 1 \\
0 & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix}
\]

\[
P_t = \begin{bmatrix}
p_{t1} & p_{t2} & \cdots & p_{tK-1} & p_{tK} \\
1 - p_{t1} & 0 & \cdots & 0 & 0 \\
0 & 1 - p_{t2} & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & 1 - p_{tK-1} & 1 - p_{tK}
\end{bmatrix}
\]

Thus, the following constraint expresses the total area in the beginning of period \( t \), for \( t = 2, \ldots, T \):

\[
s_t = (Q - P_{t-1}) x_{t-1} + P_{t-1} x_{t-1}
\]

Below, we describe the complete model.
In this model \( \delta \) is a factor used to discount monetary values and depends on the inflation rate and the size of the planning period, \( v \) is a vector with the values of the trees in each age class, standing after the last period, and \( \gamma \) is a constant penalty factor. The objective function is to maximize the expected total profit.

The following table describes the constants, data, decision variables, and constraints of the model (with its corresponding names in the Mosel code).

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha, \beta, \gamma, \delta ) (alpha,beta,gamma,delta)</td>
<td>constants of the model (read from data file)</td>
</tr>
<tr>
<td>( y ) (yield)</td>
<td>yield of selling an unit of harvested area</td>
</tr>
<tr>
<td>( v ) (v)</td>
<td>value of the trees standing after the last period</td>
</tr>
<tr>
<td>( x_t ) (x)</td>
<td>amount of harvested forest of each age class in each period</td>
</tr>
<tr>
<td>( s_t ) (s)</td>
<td>total area of forest of each age class in each period</td>
</tr>
<tr>
<td>( p_{t1} ) (p1)</td>
<td>penalty variable for not satisfying constraints ( Y_{lowerLimit} )</td>
</tr>
<tr>
<td>( p_{t2} ) (p2)</td>
<td>penalty variable for not satisfying constraints ( Y_{upperLimit} )</td>
</tr>
<tr>
<td>ProbDiscret</td>
<td>matrix of sprands representing the probabilities of fire</td>
</tr>
<tr>
<td>ValuesDiscret</td>
<td>probability discretization of the realization of each fire probability</td>
</tr>
<tr>
<td>HarvestLimit</td>
<td>constraints (1) and (2) in the model</td>
</tr>
<tr>
<td>Fix</td>
<td>auxiliar constraint that fixes ( s_1 ) variables</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>auxiliar data for fixing ( s_1 ) (initial planted area)</td>
</tr>
<tr>
<td>TotalArea</td>
<td>constraints (3) in the model</td>
</tr>
<tr>
<td>( Y_{lowerLimit} )</td>
<td>constraints (4) in the model</td>
</tr>
<tr>
<td>( Y_{upperLimit} )</td>
<td>constraints (5) in the model</td>
</tr>
</tbody>
</table>

Below, we show the Mosel code that implements the model and solves it in Xpress-SP. The matrices \( Q \) and \( P_t \) are not stored explicitly as matrices. Instead, constraints (3) of the model are generated based on the structure of both matrices using the sprand \( P \), as can be seen in the Mosel code for constraints TotalArea.

```mosel
model forest_relax
```

Illustrative examples 67 Xpress-SP Reference Manual
uses "mmsp"

parameters
EPS = 1e-5
end-parameters

declarations
T=7
K=8
MAXDISCRET=3
AGES = 1..K
index:integer
alpha,beta,delta,gama: real
ScenariosAgg,ScenariosDel: set of integer
yield, v: array (AGES) of real
ProbDiscret,ValuesDiscret: array(1..MAXDISCRET, 1..MAXDISCRET) of real
sl: array (AGES) of real
end-declarations

declarations
P: array (AGES, 2..T+1) of sprand
x: array (AGES, 1..T) of spvar
s: array (AGES, 1..(T+1)) of spvar
pl,p2: array (2..T) of spvar
HarvestLimit: array (AGES, 1..T) of splinctr
Fix: array (AGES) of splinctr
TotalArea: array (AGES, 2..(T+1)) of splinctr
YLowerLimit: array (2..T) of splinctr
YUpperLimit: array (2..T) of splinctr
end-declarations

! Read data
initializations
from 'forest.dat'
yield
v alpha beta delta gama
ProbDiscret
ValuesDiscret sl
end-initializations

! Create scenario tree
declarations
Stages = 1..(T+1)
Branches: array (1..T) of integer
Probabilities: array (1..MAXDISCRET) of real
PValues: array (1..MAXDISCRET) of real
end-declarations

spsetstages(Stages)
forall(t in 2..T+1, i in AGES) spsetstage(P(i,t), t)
Branches:= [1, 3,3,3, 3,3,3]
spcreatetree(Branches)
forall(t in 2..T+1) do
forall(i in 1..MAXDISCRET) PValues(i):= ValuesDiscret(Branches(t-1),i)
forall(i in AGES) spsetrand(P(i,t), PValues)
end-do
forall(t in 1..T) do
forall(i in 1..MAXDISCRET)
Probabilities(i):= ProbDiscret(Branches(t),i)
spsetprobcond(t+1,Probabilities)
end-do

setparam('xsp_scen_based', false)
spgentree

! Objective function
ObjFnc:= sum(t in 1..T, i in AGES) (delta^(t-1))*yield(i)*x(i,t) +
sum(i in AGES) (delta^T)*v(i)*s(i,T+1) -
gama*sum(t in 2..T) (delta^(t-1))*(pl(t) + p2(t))
! Spvars s(T+1) should be associated to stage T
forall(j in AGES, t in 1..T) HarvestLimit(j,t):= x(j,t) <= s(j,t)
forall(j in AGES) Fix(j):= s(j,1) = s1(j)
forall(i in AGES, t in 2..(T+1))
  if i=1 then
    TotalArea(i,t):= s(i,t) = sum(j in AGES) (1 -P(j,t))*x(j,t-1) +
                   sum(j in AGES) P(j,t)*s(j,t-1)
  elif i=K then
    TotalArea(i,t):= s(i,t) = -(1-P(K-1,t))*x(K-1,t-1) -
                   (1-P(K,t))*s(K-1,t-1) +
                   (1-P(K-1,t))*s(K-1,t-1) +
                   (1-P(K,t))*s(K,t-1)
  else
    TotalArea(i,t):= s(i,t) = -(1 - P(i-1,t))*x(i-1,t-1) +
                   (1 - P(i-1,t))*s(i-1,t-1)
  end-if
forall(t in 2..T) do
  YUpperLimit(t):= - sum(j in AGES) yield(j)*x(j,t) +
                  alpha*sum(j in AGES) yield(j)*x(j,t-1) <= p1(t)
  YLowerLimit(t):= sum(j in AGES) yield(j)*x(j,t) -
                  beta*sum(j in AGES) yield(j)*x(j,t-1) <= p2(t)
end-do
	forall(t in 2..T, i in AGES) do
    spsetstage(x(i,t),t)
    spsetstage(s(i,t),t)
  end-do
	forall(i in AGES) spsetstage(s(i,T+1),T)
forall(t in 2..T) do
    spsetstage(p1(t),t)
    spsetstage(p2(t),t)
end-do
setparam('xsp_verbose', true)
!
! Constraints
forall(j in AGES, t in 1..T) HarvestLimit(j,t):= x(j,t) <= s(j,t)
forall(j in AGES) Fix(j):= s(j,1) = s1(j)
forall(i in AGES, t in 2..(T+1))
  if i=1 then
    TotalArea(i,t):= s(i,t) = sum(j in AGES) (1 -P(j,t))*x(j,t-1) +
                   sum(j in AGES) P(j,t)*s(j,t-1)
  elif i=K then
    TotalArea(i,t):= s(i,t) = -(1-P(K-1,t))*x(K-1,t-1) -
                   (1-P(K,t))*s(K-1,t-1) +
                   (1-P(K-1,t))*s(K-1,t-1) +
                   (1-P(K,t))*s(K,t-1)
  else
    TotalArea(i,t):= s(i,t) = -(1 - P(i-1,t))*x(i-1,t-1) +
                   (1 - P(i-1,t))*s(i-1,t-1)
  end-if
forall(t in 2..T) do
  YUpperLimit(t):= - sum(j in AGES) yield(j)*x(j,t) +
                  alpha*sum(j in AGES) yield(j)*x(j,t-1) <= p1(t)
  YLowerLimit(t):= sum(j in AGES) yield(j)*x(j,t) -
                  beta*sum(j in AGES) yield(j)*x(j,t-1) <= p2(t)
end-do
 ! Associate Spvars to stages
forall(t in 1..T, i in AGES) do
  spsetstage(x(i,t),t)
  spsetstage(s(i,t),t)
end-do
forall(i in AGES) spsetstage(s(i,T+1),T)
forall(t in 2..T) do
  spsetstage(p1(t),t)
  spsetstage(p2(t),t)
end-do
setparam('xsp_verbose', true)
!
! Optimize
maximize(ObjFnc)
end-model

Note that an alternative structure of the scenario tree (different discretization) can be evaluated by changing the corresponding line of the Mosel code. For example, a tree with 3 branches at each stage may describe the problem more accurately and can be set by the statement Branches:= [1, 3, 3, 3, 3, 3, 3].

For the data we are using in the Appendix, the maximum number of branches (maxdiscret) in a node is 3. However, if the user provides the probabilities and the values of the corresponding sprand in the datafile, the branching structure can naturally be expanded to more than 3 branches.

5.7.1 Results

Running the model with the data provided in the Appendix gives the expected total profit of $4.23004e+007. This value is obtained for the problem when considering the structure of the scenario tree given by the statement Branches:= [1, 2, 3, 2, 2, 2, 2].

5.7.2 Analysis

The expected value version of the model has an optimal profit value of $4.21329e+007. The model is optimized with the following Mosel statement:

maximize(ObjFnc, XSP_EV+XSP_REC) ! Expected value with recourse
The value of the stochastic solution is of the order

\[
VSS = Z_R - Z_{EV} = 1.675e+005
\]

The perfect information version is solved by calling \texttt{maximize} as follows:

\[
\texttt{maximize(ObjFnc, XSP_PI)} ! \text{ Perfect information problem}
\]

This gives a profit of $4.51233e+007 and the value for the perfect information solution equal to

\[
EVPI = Z_{PI} - Z_R = 2.8229e+006
\]

In all three problems, the only nonzero first stage variable is \(x(1,8)\), that is associated to the last age class. Below, we show a table with the value of \(x(1,8)\) in the three problems:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value of (x(1,8))</th>
<th>Objective Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recourse</td>
<td>20516.9</td>
<td>4.23004e+007</td>
</tr>
<tr>
<td>Expected value</td>
<td>20738.7</td>
<td>4.21329e+007</td>
</tr>
<tr>
<td>Perfect information</td>
<td>19770.3</td>
<td>4.51233e+007</td>
</tr>
</tbody>
</table>

Suppose that after solving the model the user wants to remove scenarios that give low values for the objective function. Below, we show an additional Mosel code that makes use of the routines \texttt{speval} and \texttt{spdelscen} to inspect the objective function value (profit) associated with each scenario. The scenarios where the objective function values are less than 95% of the optimal value for the stochastic problem are removed and the problem is solved again. Using this policy, the user iteratively removes scenarios until the objective function in all scenarios is greater than 95% of the new optimal objective function value.

```mosel
Scenarios:= 1
forall(t in 1..T) Scenarios *= Branches(t)

while(true) do
  ScenariosDel:= {}
  forall(sce in 1..Scenarios| speval(ObjFnc,scen)<0.95*getobjval)
    ScenariosDel+= {scen}
    if (getsize(ScenariosDel)=0) then
      break
    end-if
  spdelscen(ScenariosDel)
  Scenarios -= getsize(ScenariosDel)
  maximize(ObjFnc)
  writeln(getobjval)
end-do
```

The variable \texttt{ScenariosDel} is a set of integers that specifies the scenarios that will be deleted. The table below shows the objective function value for the stochastic problem at each iteration of this procedure, until there are no such scenarios to be removed.

Iteration 0 refers to the objective function value of the original stochastic problem. The original problem had 729 scenarios and the final problem solved at iteration 6 had 675 scenarios. The scenario tree for this problem is generated by using the branching scheme \texttt{Branches:= [1, 3, 3, 3, 3, 3, 3]}.

It is also possible to make a similar exercise, aggregating scenarios that have low probability by using the routines \texttt{spget.scenprob} and \texttt{spaggregate}. The following code aggregates scenarios that have probability less than 0.1% and are contiguous in the scenario tree, with the same parent node. After aggregation, we end up with 480 scenarios and expected total profit $4.2484e+007.
Table 5.4: Expected profit after scenario deletion

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Expected profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.22529e+007</td>
</tr>
<tr>
<td>1</td>
<td>4.22907e+007</td>
</tr>
<tr>
<td>2</td>
<td>4.22990e+007</td>
</tr>
<tr>
<td>3</td>
<td>4.23034e+007</td>
</tr>
<tr>
<td>4</td>
<td>4.23041e+007</td>
</tr>
<tr>
<td>5</td>
<td>4.23072e+007</td>
</tr>
<tr>
<td>6</td>
<td>4.23096e+007</td>
</tr>
</tbody>
</table>

Scenarios:= 1
forall(t in 1..T) Scenarios *= Branches(t)

ScenariosAgg:= {}
forall(scen in 1..Scenarios | spgetprobscen(scen)<0.001)
  ScenariosAgg+= {scen}

scen:= getsize(ScenariosAgg)
AggOccurred:= false
while(scen >= 1) do
  siblings:= {ScenariosAgg(scen)}
  forall(j in 1..(MAXDISCRET-1) | scen-j>0 ) do
    if (ceil(ScenariosAgg(scen-j)/Branches(T)) = ceil(ScenariosAgg(scen)/Branches(T))) then
      siblings+= {ScenariosAgg(scen-j)}
    else
      break
    end-if
  end-do
  if (getsize(siblings)>1) then
    writeln('aggregating ', siblings)
    spaggregate(siblings)
    AggOccurred:= true
  end-if
  Scenarios -= getsize(siblings) + 1
  scen -= getsize(siblings)
end-do

maximize(ObjFnc)
writeln(getobjval)

The set ScenariosAgg is a set of integers that specifies the scenarios with probability less than 0.1%. From ScenariosAgg, we select the scenarios to be aggregated by checking if they have the same parent in stage 7. This task is done by using the information on the number of branches in stage 7:

ceil(ScenariosAgg(scen)/Branches(T))

Below, we show a table that presents the differences between the node-based version and the scenario-based version of this model, in terms of number of rows and columns. The values are shown before and after pre-solving takes place, for the problem with 729 scenarios.

Table 5.5:

<table>
<thead>
<tr>
<th></th>
<th>Before presolve</th>
<th>After presolve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rows</td>
<td>Columns</td>
</tr>
<tr>
<td>Node Based</td>
<td>12400</td>
<td>8512</td>
</tr>
<tr>
<td>Scenario Based</td>
<td>183944</td>
<td>96228</td>
</tr>
</tbody>
</table>

5.7.3 Summary

In this section, we have just shown some of the features, embedded in Xpress-SP, that allow the
user to optimize stochastic problems and perform further analysis in an easy and customizable way: by inspecting variables in particular scenarios, aggregating/deleting scenarios, obtaining estimates from the expected value and perfect information problems, and choosing among different ways of expanding the problem. These ideas play an important role in providing insight into the problem and simplifying or strengthening its formulation.

5.8 Assets and liabilities management for pension funds

Assets and liabilities management (ALM) problems are one of the most important classes of problems arising in the financial area. Typically the goal of these problems is to find an optimal strategy for allocating resources in order to obtain a profit that is sufficient to cover a certain type of liability. These problems often involve uncertain parameters; for instance, the returns received from different investments usually depend on unknown future market conditions. The amounts of liabilities that need to be paid at certain time can also depend on various future factors. Therefore, in many cases, applying stochastic programming techniques is crucial for making strategic decisions that efficiently handle the risk of possible losses.

We consider one of the practically important instances of these problems — assets and liabilities management for pension funds. A typical pension fund has members that can be separated into two classes. The first class is ‘active members’, i.e., employees paying a certain fraction of their wages to the pension fund each month. The second class is ‘non-active’ members, i.e., people who have retired and receive payments from the pension fund. At every time period the pension fund should have enough assets to be able to cover all the payments (liabilities). The problem the pension fund manager is faced with is minimizing the contribution rate (fraction of wages) paid by the active members while being able to cover the liabilities. The setup of the problem aims to ensure that the percentage of wages that the employees pay is small enough to attract new pension fund members. To cover the liabilities, assets owned by the fund are invested into stocks, whose returns are uncertain. Moreover, the amount of liabilities is also uncertain, since it depends on the inflation rate and demographic factors. Stochastic programming techniques allow one to expand the deterministic formulation of the model by using scenario data for uncertain parameters.

First, we present the formal setup of the model. These are its parameters, decision variables, and the deterministic formulation (assuming that all the parameters are known and no uncertainty is involved). Here we consider the one-period model, i.e., there is only one planning period, where the investment decision is made at the initial moment in order to cover the liabilities at the end of the period.

5.8.1 Parameters of the model

$A_0$ – the initial value of all assets owned by the fund.
$W_0$ – the total amount of wages of the active members at the initial moment.
$l_0$ – the payments made by the fund at the initial moment.
$r_n$ – the rate of return of the stock number $n$, $n = 1, \ldots, N$.
$L$ – the liability measure that should be exceeded by the total value of assets at the end of the period.
$\psi$ – the parameter used to control the behavior of the fund. We want the total return to exceed the liabilities by a certain amount, for instance, if $\psi = 1.3$ then we want the total return to be at least 30% higher than the liabilities.

5.8.2 Decision variables

$x_n$ – the total amount of money invested in the stock number $n$, $n = 1, \ldots, N$.
$y$ – the contribution rate (fraction of wages) that is added to the initial assets at the initial time moment.
5.8.3 Deterministic problem formulation

\[ Z = \min_{y \in \mathbb{R}, x \in \mathbb{R}^N} y \]
\[ \sum_{n=1}^{N} x_n = A_0 - l_0 + W_0 y \]
\[ \sum_{n=1}^{N} (1 + r_n) x_n \geq \psi L \]
\[ x_n \geq 0 \quad \forall n \in 1..N \]

5.8.4 Description

The first constraint is the investment balance: the total money invested in the stocks \( x_n \) should be equal to the total assets available at the beginning of the period, i.e., the initial assets plus the fraction of wages paid by active members minus payments made at the beginning of the period.

The second constraint expresses the fact that the total return at the end of the period must exceed the amount of liabilities (multiplied by the control parameter \( \psi \)).

The third set of constraints is the non-negativity constraints on \( x_n \) that means that we don’t allow any 'short' positions or borrowing. Note that we don’t impose any restrictions on \( y \), we allow it to be negative. If \( y \) is negative, it means that the fund can cover all the liabilities without taking money from active members and even make additional payments to them.

5.8.5 Stochastic problem formulation

We now consider the ALM problem with \( N = 12 \) stocks and \( S = 5000 \) scenarios corresponding to the realizations of the returns and liabilities. We create the scenario tree based on the real-life data representing returns of 12 stocks, as well as the amount of liabilities. This tree has two stages and 5000 nodes at the second stage. Certain realizations of the returns and liabilities correspond to each node at the second stage. The probabilities of all realizations are assumed to be the same and equal to \( P = 1 / S \).

The extended two-stage formulation of the problem can be written as follows:

\[ Z = \min_{y \in \mathbb{R}, x \in \mathbb{R}^N} y \]
\[ \sum_{n=1}^{N} x_n = A_0 - l_0 + W_0 y \]
\[ \sum_{n=1}^{N} (1 + r_{n}^s) x_n \geq \psi L^s \quad \forall s \in 1..S \]
\[ x_n \geq 0 \quad \forall n \in 1..N \]

5.8.6 Mosel implementation

Using stochastic modeling tools in Mosel, one can create the scenario tree representing stochastic parameters of the model and specify the constraints of the deterministic model then the extended (stochastic) problem formulation is generated automatically. The Mosel implementation of the stochastic model is presented below.

```
model ALM_sp
uses 'mmsp' ! Use the stochastic programming module
parameters
psi=1.3
end-parameters
declarations
Stocks=1..12 ! Set of stocks
Initial_Asset,A: real ! Initial asset
Initial_payment,l: real ! Payments at the initial time
Total_wages,W: real ! Total amount of wages of active members of the fund
end-declarations
```

Illustrative examples 73 Xpress-SP Reference Manual
! Initial (deterministic) parameters
initializations from 'ALM.dat'
Intial_Asset Initial_payment Total_wages
end-initializations

A:=1 ! Scaled value of initial assets
l:=Initial_payment/Intial_Asset ! Scaled value of initial payments
W:=Total_wages/Intial_Asset ! Scaled value of wages

declarations
Stages = {"First","Second") ! Two stages - one-period model
! Random parameters
return: array(Stocks) of sprand ! Uncertain returns of the stocks
L: sprand ! Uncertain liabilities
! Decision variables
x: array(Stocks) of spvar ! Amount invested into each stock
y: spvar ! % of wages paid by active members
! Model objective and constraints
fraction: splinctr
inv_balance: splinctr
minreturn: splinctr
end-declarations

! Allow contribution rate to be both positive and negative
y is_free

! Setting stages
spsetstages(Stages)
forall(p in Stocks) spsetstage(return(p),"Second")
spsetstage(L,"Second")

! Scenarios declaration and generating scenario tree
declarations
S = 500
Scen = 1..S
Values: array(Scen, Stocks) of real
Liab: array(Scen) of real
end-declarations

cpcreatetree(S)
forall(s in Scen) prob(s):=1/S
spsetprobcond(2,prob)

initializations from 'returns.dat'
Values
end-initializations

initializations from 'Liabilities.dat'
Liab
end-initializations

forall(p in Stocks,s in Scen) spsetrandatnode(return(p),s,Values(s,p))
forall(s in Scen) spsetrandatnode(L,s,Liab(s)/Intial_Asset)
spgentree

! Model definition
fraction:= y
inv_balance:=sum(p in Stocks)1*x(p) = A - l + W*y
minreturn:=sum(p in Stocks)(1+return(p))*x(p) >= psi*L

! Assigning variables to stages
forall(p in Stocks) spsetstage(x(p),"First")
spsetstage(y,"First")

! Optimizing
minimize(fraction)

! Solution output
writeln("psi = ", psi)
writeln("contribution rate = ", getobjval)
forall(p in Stocks) writeln("x(" , p , ") = ", getsol(x(p)))
Note that for convenience the monetary parameters of the model are scaled, i.e., divided by the amount of the initial assets $A_0$. The description of the data files `data.dat` and `data_-L.dat` used for generating the scenario tree can be found in the Appendix. The expected value ($Z_{EV}$) and perfect information ($Z_{PI}$) solutions can be found by running this program after modifying the line `minimize(fraction)` by `minimize(fraction, XSP_EV)` and `minimize(fraction, XSP_PI)` respectively.

### 5.8.7 Analysis of the results

Table B.1 (see Appendix) presents the values of the objective function (i.e., contribution rate) for the original stochastic problem ($Z^*$), and for the expected value and perfect information problems, for different values of the control parameter $\psi$. The differences between these solutions are also calculated.

An interesting observation is that all these solutions, as well as the differences $Z^* - Z_{EV}$ and $Z^* - Z_{PI}$ are linear functions of $\psi$ (see Figure 5.12). Linear dependency of $Z^*$, $Z_{EV}$ and $Z_{PI}$ of $\psi$ can be easily proven theoretically, using LP sensitivity analysis techniques. Since $Z^*$, $Z_{EV}$ and $Z_{PI}$ are linear functions of $\psi$, the differences $Z^* - Z_{EV}$ and $Z^* - Z_{PI}$ representing the ‘value’ of deterministic solution and perfect information are also linear with respect to $\psi$. Note that the optimal stochastic solution is much larger than the corresponding expected value and perfect information solutions, which means that one is forced to take more conservative decisions (i.e., spend more money) if uncertainty is taken into account. Also, as $\psi$ increases, the values $Z^* - Z_{EV}$ and $Z^* - Z_{PI}$ also increase, which means that the stochastic solution becomes more ‘expensive’ if a greater amount of liabilities needs to be covered.

It is also important to look at the optimal investment strategy found from solving the stochastic problem. As one can see from Table B.2 (see Appendix), the relative structure of the investments remains the same for all considered $\psi$’s, however, the amounts invested into each stock
Another issue that we address here is how the structure of the scenario dataset affects the optimal solution of the stochastic problem. For this purpose, we calculated the mean $\mu_n$ and the standard deviation $\sigma_n$ of the return of each considered stock, over 5000 scenarios. Then we delete all scenarios where the value of the return of at least one stock lies outside the interval $[\mu_n - m\sigma_n, \mu_n + m\sigma_n]$, for some value of $m$. This procedure allows one to reduce the variability of the scenarios by excluding ‘extreme’ values of the returns, which can be both positive and negative. Deletion of a set of scenarios can be performed using the function spdelscen(). By changing the parameter $m$, we control the length of the interval and establish the bounds for possible values of the returns. The purpose of this procedure is to compare the solution of the initial problem and the solutions corresponding to the problems with fewer scenarios having smaller variability.

As one can intuitively predict, the optimal solution will be ‘cheaper’ if the length of this interval is small, and the contribution rate will increase as the interval length grows, since the minimum return constraint needs to be satisfied for all scenarios, including realizations with extreme unfavorable returns. As a limiting case, when $m$ is large enough to include all the considered set of scenarios, the solution of the original stochastic problem is obtained. These results are summarized in Table 8.

Figure 5.13 shows the plot of the optimal solution value with respect to the number of scenarios considered. As one can see, when the number of scenarios is very close to 5000, the optimal solution is almost the same as the ‘true’ solution corresponding to the problem with all 5000 scenarios; however, as the number of scenarios decreases, the optimal objective value drops down. This happens because the ‘extreme’ scenarios that affect the solution are deleted from the scenario tree, resulting in a less expensive solution. The difference between the true solution and the solution obtained after the deletion of scenarios remains relatively small as long as the number of remaining scenarios is large enough ($\approx 2000$). This sufficiently reflects the variability of the returns; however, this difference rapidly increases if more scenarios are deleted. In fact, if we leave only the scenarios where all the returns lie in the interval $[\mu_n - \sigma_n$.
, \( \mu_n + \sigma_n \), the optimal solution value is almost twice smaller than in the case of the full set of 5000 scenarios. These facts show that the appropriate choice of the scenarios dataset is important for obtaining the solution of the stochastic problem that would properly reflect the uncertainty.

5.9 Assemble to order system

Refer to Section 5 in the paper [DVVT05].

5.10 Options contract in supply chains

Refer to Section 6 in the paper [DVVT05]

5.11 Power management in a hydro-thermal system

Stochastic Programming has been used extensively in the energy sector for various purposes such as planning of capacity expansion, unit commitment, making pricing decision in the electricity market, etc. The following example has been inspired from the work of Gröwe-Kuska et al. (see [GKKN+02]). Here we consider a hydro-thermal system in which hourly decisions pertaining to operational levels of the units need to be made under uncertain demand load.

5.11.1 Description

There are \( T \) time periods. \( I \) and \( J \) are the numbers of thermal and hydro units, respectively. For a thermal unit \( i \) in period \( t \), \( u_{i,t} \in \{0,1\} \) is its commitment (1 if on, 0 if off) and \( p_{i,t} \) its production, with \( p_{i,t} = 0 \) if \( u_{i,t} = 0 \), \( p_{i,t} \in [P_{\min,i,t}, P_{\max,i,t}] \) if \( u_{i,t} = 1 \). Additionally there are minimum up/down-time requirements: when unit \( i \) is switched on (off) it must remain on (off) for at least \( \tau_{\text{up},i} \) (\( \tau_{\text{dn},i} \)) periods. For a hydro plant \( j \), \( s_{j,t} \) and \( w_{j,t} \) are its generation and pumping levels in period \( t \) with upper bounds \( S_{\max,j,t} \) and \( W_{\max,j,t} \) respectively, and \( l_{j,t} \) is the storage volume in the upper dam at the end of period \( t \) with upper bound \( L_{\max,j,t} \). The water balance relates \( l_{j,t} \), with \( l_{j,t-1} \), \( s_{j,t} \), \( w_{j,t} \) and water inflow \( \gamma_{j,t} \) using the pumping efficiency \( \eta_{j} \). The initial and final volumes are specified by \( l_{\text{in},j} \) and \( l_{\text{end},j} \).

The basic system requirement is to meet the electric load. Another important requirement is the spinning reserve constraint. To maintain reliability (compensate sudden load peaks or unforeseen outages of units) the total committed capacity should exceed the load in every period by a certain amount (e.g., a fraction of the demand). The load and the spinning reserve during period \( t \) are denoted by \( d_{t} \) and \( r_{t} \), respectively.

Efficient operation of pumped storage hydro plants exploits daily cycles of load curves by generating during peak load periods and pumping during off-peak periods. Since the operating costs of hydro plants are usually negligible, the total system cost is given by the sum of startup and operating costs of all thermal units over the whole scheduling horizon. The fuel cost \( C_{i,t} \) for operating thermal unit \( i \) during period \( t \) has the form \( C_{i,t} = \max_{c_{i} \in \{1,\ldots,L\}} a_{i,c_{i,t}} \cdot p_{i,t} + b_{i,c_{i,t}} \cdot u_{i,t} \). The startup cost of unit \( i \) depends on its downtime; it may vary from a maximum cold start value to a much smaller value when the unit is still relatively close to its operating temperature. This is modeled by the startup cost \( S_{i,t} = \max_{c_{i} \in \{1,\ldots,L\}} a_{i,c_{i,t}} \cdot u_{i,t} - \sum_{k \in \{1,\ldots,T\}} u_{i,k-\tau} \) where \( 0 = c_{0} < \ldots < c_{T} \) are fixed cost coefficients. \( T \) is the cool-down time of unit \( i \), \( c_{i,\tau} \) is its maximum cold-start cost, \( u_{i,\tau} \)

\( \forall \tau \in \{\tau_{\text{ini}}, \ldots, 0\} \) are given initial values, where \( \tau_{\text{ini}} = \max_{c_{i} \in \{1,\ldots,L\}} \{c_{i}, 1 - \tau_{\text{up},i}^{\tau}, \tau_{\text{dn},i}^{\tau} - 1\} \).
5.11.2 Model

\[ \min \ E[\sum_{t} (C_{i,t} + S_{i,t})] \]

\[ S1 \]

\[ p_{i,t}^{\min} u_{i,t} \leq p_{i,t} \leq p_{i,t}^{\max} u_{i,t} \]

\[ u_{i,t} - u_{i,t-1} \leq u_{i,t}, \quad \forall t - \tau_{i}^{up} < \tau < t \]

\[ u_{i,t-1} - u_{i,t} \leq 1 - u_{i,t}, \quad \forall t - \tau_{i}^{dn} < \tau < t \]

\[ 0 \leq s_{j,t} \leq S_{j,t}^{\max}, \quad 0 \leq w_{j,t} \leq W_{j,t}^{\max}, \quad 0 \leq l_{j,t} \leq L_{j,t}^{\max} \]

\[ l_{j,t} = l_{j,t-1} - s_{j,t} + \eta_{j} w_{j,t} + \gamma_{j,t} \]

\[ l_{j,0} = l_{j}^{\infty}, \quad l_{j,T} = l_{j}^{\text{end}} \]

\[ \sum_{i} p_{i,t} + \sum_{j} (s_{j,t} - w_{j,t}) \geq \tilde{d}_{t} \]

\[ \sum_{i} (u_{i,t} - p_{i,t}^{\max} - p_{i,t}) \geq r_{t} \]

5.11.3 Uncertainty

The uncertainty in the model—demand in each period—is dealt with as follows:

1. We begin by aggregation of periods. As the number of periods increases, the possibilities of realization of demand increase exponentially. From the stochastic optimization point of view, some of the consecutive periods may be aggregated together to form a stage, e.g., periods \{1, \ldots, t_{1}\} may be set to stage 1, periods \{t_{1}+1, \ldots, t_{2}\} to stage 2, ..., and periods \{t_{N}+1, \ldots, T\} to stage \(N\) as shown in the following Figure 5.14.

2. Demand, being random, may be identified based on a time series or regression model. Here we simply assume the demand process to be \(\tilde{d}_{t} = \alpha_{t} + \beta_{t} \tilde{d}_{t-1} + \lambda \tilde{\xi}_{t}\), where \(\alpha_{t}, \beta_{t}, \lambda\) are given, and \(\tilde{\xi}_{t} \sim Uniform(\xi_{\min}, \xi_{\max})\), and \(\tilde{\xi}_{t} \sim Normal(0, 1)\)

3. Based on the demand process, we determine an initial structure of the scenario tree. We compute values of demands in various scenarios using the sample means and standard deviations of the simulated scenarios by running \(M\) simulations. If \(d_{t}^{m}\) is the value of demand in the \(m^{th}\) run then estimated

(a) mean demand:
Figure 5.15: Binary scenario tree with $2^{N-1}$ scenarios

\[
\mu_t = \sum_{m=1}^{M} d_{mt}^m,
\]

and

(b) variance in demand:

\[
\sigma_t^2 = \frac{\sum_{m=1}^{M} (d_{mt}^m - \mu_t)^2}{M-1}
\]

4. Next, we build a stable and small scenario tree using the estimates and aggregated stages as follows:

(a) Re-model the demand process as

\[
d_t = \mu_t + \sum_{t'=2}^{t} \delta_{Stg(t')}, \left(\frac{\sigma_{t'}^2}{2^{t'-1}}\right)^{1/2},
\]

where $Stg(t)$ is the aggregated stage where period $t$ belongs to, and $\delta_2, \ldots, \delta_N \sim \{1 \text{ w.p.} 0.5, -1 \text{ w.p.} 0.5\}$

(b) Generate an exhaustive binary tree based on possible realizations of $\delta_2, \ldots, \delta_N$ as shown below.

5. As the number of aggregated stages increase, the number of scenarios may increase rapidly, therefore we may reduce the scenario tree using the following schema.

(a) Define the distance between vectors $v^i$ and $v^j$ as $c_h(v^i, v^j) = \max \{1, ||v^i||^{h-1}, ||v^j||^{h-1}\}$.

(b) Let realizations of $\delta = [\delta_2, \ldots, \delta_N]$ in scenario $k$ be vector $\delta^k$. Given $S$ scenarios with probabilities $P_1, \ldots, P_S$, we delete a scenario $k = \arg\min_{l \in \{1, \ldots, S\}} \{ P_l, \min_{s \in \{1, \ldots, S\} : s \neq l} c_h(\delta^l, \delta^s) \}$. Then, roughly speaking, deletion of a scenario $k$ occurs when it is close to another scenario $s$ as measured by the distance or where its probability is small. The reduced scenario tree has one less scenario; scenario $k$ is deleted, and $P_s$ is incremented by $P_k$.

This step is repeated until the final number of scenarios reaches a prescribed number $S'$ that is reasonably small to represent the uncertainty, and would yield sufficiently accurate solution to the problem.

(c) Next, we decouple $S'$ scenarios by:

i. Creating a scenario tree with $S'$ branches from the node in the $1^{st}$ stage, and 1 branch per node from each node in other stages.

ii. Setting the realized values of $\delta^{l+1}$, and revised probabilities of scenarios. The scenario tree would look as shown in the following Figure 5.16.

(d) Many realizations of $\delta$'s until certain stages are the same, hence we restructure the reduced scenario tree by clubbing together all the realizations that are equal as follows:

i. Define $node_{t,n}$ for the $n^{th}$ node in the $t^{th}$ stage of the scenario tree. Let $PossScen_{t,n}$ be set of scenarios that branches off from $node_{t,n}$ that have same realized values of $\delta_{t+1}$

ii. Find all such possible sets of the $PossScen_{t,n}$'s corresponding to the realizations of $\delta_{t+1}$
iii. Each of the set $\text{PossScen}_{t,n}$ forms a new child of node $t,n$ with the conditional probability: $\sum_{s\in\text{PossScen}_{t,n}} P_s / P_{t,n}$, where $P_{t,n}$ is the unconditional probability of occurrence of node $t,n$. While keeping track of number of children of node $t,n$, number of nodes in stage $t$, and unconditional probability $P_{t,n}$ of node $t,n$, incrementally build a scenario tree. The re-structured scenario tree would look as follows.

5.11.4 Optimization

The optimization model is built in Xpress-SP by associating all decision variables in period $t$ to $\text{Stg}(t)$. The scenario tree is built by running simulation, building a binary scenario tree, and reducing it by deleting few scenarios as described in previous section. The model excerpt is shown below.

```plaintext
declarations
 p,S,C: array(ThermalUnits,TimeBlocks) of spvar
 u: array(ThermalUnits,(tau_ini..0)+TimeBlocks) of spvar
 s,w,l: array(HydroUnits,TimeBlocks) of spvar
 lam: array(ThermalUnits,TimeBlocks,1..L) of spvar
 TotalCost: splinctr
 UBprod: array(ThermalUnits,TimeBlocks) of splinctr
 LBprod: array(ThermalUnits,TimeBlocks) of splinctr
 UBstorageBal: array(HydroUnits,TimeBlocks) of splinctr
 Minuptime,Minowntime: dynamic array(ThermalUnits,range,TimeBlocks) of splinctr
 Load: array(TimeBlocks) of splinctr
 SpinningReserve: array(TimeBlocks) of splinctr
 ProdLevel,FuelCost,SumToOne: array(ThermalUnits,TimeBlocks) of splinctr
 Prodtn: dynamic array(ThermalUnits,TimeBlocks) of splinctr
 StartupCost: array(ThermalUnits,range,TimeBlocks) of splinctr
 end-declarations

forall(i in ThermalUnits, t in (tau_ini..0)+TimeBlocks) u(i,t) is_binary

forall(t in tau_ini..0,i in ThermalUnits) do
    spsetstage(u(i,t),1)
    u(i,t)=0
end-do

forall(i in ThermalUnits,t in TimeBlocks)
    UBprod[i,t] := p[i,t] <= P_max(i) * u(i,t)
```

Illustrative examples 80 Xpress-SP Reference Manual
forall(i in ThermalUnits, t in TimeBlocks)
LBprod(i,t):=p(i,t)>=P_min(i)*u(i,t)
forall(j in HydroUnits, t in TimeBlocks)
s(j,t)<=S_max(j)
forall(j in HydroUnits, t in TimeBlocks)
w(j,t)<=W_max(j)
forall(j in HydroUnits, t in TimeBlocks)
l(j,t)<=SL_max(j)
forall(j in HydroUnits, t in TimeBlocks)
if t=1 then
UBstoreageBal(j,t):= l(j,t)=l_in(j)-s(j,t)+eta(j)*w(j,t)+gamma(j,t)
elif t=T then
UBstoreageBal(j,t):= l_end(j)=l(j,t-1)-s(j,t)+eta(j)*w(j,t)+gamma(j,t)
else
UBstoreageBal(j,t):= l(j,t)=l(j,t-1)-s(j,t)+eta(j)*w(j,t)+gamma(j,t)
end-if
forall(i in ThermalUnits, t in TimeBlocks, tau in (t-tau_up(i)+1)..(t-1) | t-tau_up(i)+1<=t-1 and tau-1>=tau_ini)
Minuptime(i,tau,t):=u(i,tau)-u(i,tau-1)<=u(i,t)
forall(i in ThermalUnits, t in TimeBlocks, tau in (t-tau_dn(i)+1)..(t-1) | t-tau_dn(i)+1<=t-1 and tau-1>=tau_ini)
Mindowntime(i,tau,t):=u(i,tau-1)-u(i,tau)<=1-u(i,t)
forall(t in TimeBlocks)
Load(t):=sum(i in ThermalUnits) p(i,t)+
sum(j in HydroUnits) (s(j,t)-w(j,t))>=d(t)
forall(t in TimeBlocks) SpinningReserve(t):=
sum(i in ThermalUnits) (P_max(i)*u(i,t)-p(i,t))>=r(t)
forall(i in ThermalUnits, t in TimeBlocks) do
ProdLevel(i,t):=p(i,t)=
sum(l in 1..L) prodt(i,t,l_)*lam(i,t,l_)
FuelCost(i,t):=C(i,t)=sum(l in 1..L) cost(i,t,l_)*lam(i,t,l_)
SumToOne(i,t):=sum(l in 1..L) lam(i,t,l_)=u(i,t)
Prodtn(i,t):=sum(l in 1..L) prodt(i,t,l_)*lam(i,t,l_)
Prodtn(i,t) is sos2
end-do
forall(i in ThermalUnits, t in TimeBlocks, tau in 0..tau_c(i))
StartupCost(i,tau,t):=
S(i,t)>=c(i,tau)*(u(i,t)-sum(k in 1..tau) u(i,t-k))
5.11.5 Implementation

The model is implemented for 24 time periods (1 day) aggregated into 5 stages, 2 hydro units and 3 thermal units. A visual interface is built to visualize the demand and optimal solutions in various scenarios as shown in the following figure.
II. Reference manual
Chapter 6

Types

6.1 sprand

SP random element for describing underlying uncertainty

6.2 sprandexp

SP random expression built using sprands and reals

6.3 spvar

SP decision variable

6.4 splinctr

SP linear constraint built using reals, sprand, and sprandexp
Chapter 7
Functions and procedures

abs, ceil, floor  non-linear stochastic functions  p. 124
getdual, getdualatnode, getdualinscen  obtain dual  p. 104
getobjval  obtain objective value  p. 100
getcost, getcostatnode, getcostinscen  obtain reduced cost  p. 103
getslack, getslackatnode, getslackinscen  obtain slack  p. 105
getsol, getsolatnode, getsolinscen  obtain solution  p. 102
maximize, minimize  optimize current SP  p. 98
maximum, minimum  get max/min of stochastic enetities  p. 123
spaddchildren  create scenario tree structure node by node  p. 94
spaggregated  aggregate scenario(s)  p. 120
spcreattree  create scenario tree structure  p. 92
spdelscen  delete scenario(s)  p. 119
speval  evaluate stochastic entity  p. 106
spexportprob  export stocahstic problem  p. 122
spfix  fix a stochastic variable  p. 107
spgenexh  generate exhaustive scenario tree  p. 91
spgentree  generate scenario tree  p. 97
spgetchild  get a childe of a node  p. 118
spgetchildrencount  get number of children of a node  p. 117
spgetnodecount  get number of nodes  p. 112
spgetobjconst  constant term in the objective  p. 101
spgetparent  get parent node number  p. 116
spgetprobcond  get conditional probability  p. 113
spgetprobscen  get probability of a scenario  p. 115
spgetprobuncond  get unconditional probability  p. 114
spgetscencount  get scenario count  p. 111
spif  stochastic conditional functions  p. 125
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>spprinttree</td>
<td>print current scenario tree</td>
<td>121</td>
</tr>
<tr>
<td>spsetdist</td>
<td>set discretized distribution of stochastic random element</td>
<td>89</td>
</tr>
<tr>
<td>spsethidden</td>
<td>hide or unhide a stochastic linear constraint</td>
<td>109</td>
</tr>
<tr>
<td>spsetname</td>
<td>set name of stochastic entity</td>
<td>126</td>
</tr>
<tr>
<td>spsetprobcond, spsetrand</td>
<td>set conditional probabilities</td>
<td>93</td>
</tr>
<tr>
<td>spsetprobcondatnode, spsetrandatnode</td>
<td>set conditional probabilities</td>
<td>96</td>
</tr>
<tr>
<td>spsetprobscen</td>
<td>set scenario probabilities</td>
<td>95</td>
</tr>
<tr>
<td>spsetstage</td>
<td>associate stochastic entities with stages</td>
<td>88</td>
</tr>
<tr>
<td>spsetstages</td>
<td>set stages of stochastic problem</td>
<td>87</td>
</tr>
<tr>
<td>spsettype</td>
<td>set type of a stochastic constraint</td>
<td>110</td>
</tr>
<tr>
<td>spunfix</td>
<td>unfix a stochastic variable</td>
<td>108</td>
</tr>
</tbody>
</table>
**spsetstages**

**Purpose**
Used for setting the stages of a stochastic program.

**Synopsis**

spsetstages(Stages: range)
spsetstages(Stages: set of integer)
spsetstages(Stages: set of string)

**Example**

```plaintext
declarations
T=2
Stages=1..T
end-declarations
spsetstages(Stages)
```

**Further information**

1. This should be the first procedure to be called.
2. Once the stages are set, they cannot be changed.

**Related topics**

- spsetstage
**spsetstage**

**Purpose**

It is used for setting the stages of a sprand or a spvar.

**Synopsis**

- `spsetstage(x: spvar, t: integer)`
- `spsetstage(rn: sprand, t: integer)`
- `spsetstage(x: spvar, t: string)`
- `spsetstage(rn: sprand, t: string)`

**Arguments**

- `x` stochastic variable
- `rn` stochastic random element
- `t` stage belonging to the stochastic stage set which was set by calling `spsetstages()`

**Further information**

1. This procedure should be called only after calling `spsetstages()`.
2. The stochastic entity may be dissociated from SP stages by setting its stage to `XSP_STAGE_ZERO`.

**Related topics**

- `spsetstages`
**spsetdist**

**Purpose**
It is used for setting discretized distribution of sprand.

**Synopsis**

- `spsetdist(rn: sprand, val: array() of real, prob: array() of real)`
- `spsetdist(rns: set of sprand, val: array() of real, prob: array() of real)`

**Arguments**

- **rn**: independent stochastic random element
- **rns**: jointly independent stochastic random elements
- **val**: discrete values rn(s) may assume
- **prob**: probabilities with which rn(s) assumes discrete values

**Example**

**Ex 1**: Following code shows how to set an sprand's independent distribution

```plaintext
declarations
val,prob:array(1..3) of real
end-declarations
val:=[10,20,30];prob:=[0.2,0.5,0.3]
spsetdist(rn,val,prob)
```

**Ex 2**: Following code shows how to set joint distribution of set of sprands

```plaintext
! {return(‘Stock’),return(‘Bond’),return(‘Cash’)}= [1.20, 1.08, 1.02] w.p 0.3
! = [0.90, 1.03, 1.00] w.p 0.7

declarations
Asset={'Stock','Bond','Cash'}
return: array(Asset) of sprand
rns: set of sprand
val:array(Asset,1..2) of real
prob:array(1..2) of real
end-declarations
rns:=union(i in Asset) {return(i)}; ! {return(‘Stock’),return(‘Bond’),return(‘Cash’)}
val:=[1.20, 0.90, 1.08, 1.03, 1.02, 1.00]; ! Stock, Bond, Cash
prob:=[0.3, 0.7]; ! up branch, down branch
spsetdist(rns,val,prob)
```

**Further information**

1. If rn belongs to first stage then by default it is equal to 0. Its value may however be changed by setting its distribution corresponding to a discrete value with probability 1.0.

2. If some sprands are jointly distributed then their discretized distributions can be passed using the overloaded procedures.

3. Tree can be generated by calling spgenexh().
Related topics
  spgenexh
spgenexh

Purpose
It is used for generating exhaustive scenario tree based on discretized distributions of stochastic random elements.

Synopsis
spgenexh

Further information
This procedure should be called after setting discretized distributions of all stochastic random elements.

Related topics
spsetdist
spcreatetree

Purpose
It is used for creating the structure for scenario tree without setting the values of random elements, or the probabilities in the tree.

Synopsis
spcreatetree(sb: integer)
spcreatetree(br: array(range) of integer)
spcreatetree(nd: array(range,range) of integer)

Arguments
sb number of branches from each node in a symmetric scenario tree
br number of branches from nodes in each stage of a scenario tree
nd node number of node in each scenario at each stage

Example
Ex 1:

spcreatetree(2) !create binary tree

Ex 2:

branches:=[2,3]
spcreatetree(branches) ! create a 3-stage tree with 2 branches from
! first stage node,
! and 3 branches from second stage nodes.

Ex 3:

NodeNum:=[
  1,1,1,
  1,1,2,
  1,2,3,
  1,2,4];
spcreatetree(NodeNum) ! create a 3-stage binary tree explicitly

Further information
1. All sprands must have been associated with stages before creating scenario tree.
2. Tree can be generated by setting values of sprands in the tree and probabilities of visiting nodes or scenarios, followed by calling spgentree().

Related topics
spsetprobscen, spsetprobcond, spsetprobcondatnode, spsetrand, spsetrandatnode, spaddchildren, spgentree, spprinttree
**spsetprobcond, spsetrand**

**Purpose**

It is used for setting values of random elements and their conditional probabilities at all the nodes in a stage.

**Synopsis**

```
spsetprobcond(p: array(range) of real)
spsetprobcond(t: integer, p: array(range) of real)
spsetprobcond(rn: sprand, v: array(range) of real)
```

**Arguments**

- `t` stage number belonging to 1..T, where T is number of stages
- `p` conditional probability distribution of visiting nodes in the t<sup>th</sup> stage
- `rn` stochastic random element
- `v` realized values of `rn` at nodes in the t<sup>th</sup> stage, where t is the stage number of `rn`

**Example**

Ex 1:

```plaintext
! rn(2),...,rn(T) distributed with {10 w.p 0.3, 20 w.p 0.7}
spcreatetree(2) !create binary tree
val:=[10,20]
forall(t in 2..T) spsetrand(rn(t),val)
prob:=[0.3,0.7] !conditional probabilities
spsetprobcond(3,prob)
```

Ex 2:

```plaintext
branches:=[2,4] ! 8 scenarios
spcreatetree(branches) ! create a 3-stage tree with 2 branches from
  ! first stage node,  
  ! and 4 branches from second nodes.
val1to2:=[50,100] !realized value of rn(2) at nodes in stage 2
spsetrand(rn(2),val1to2)
prob1to2:=[0.5,0.5] ! conditional probabilities of visting stage 2
  ! nodes from stage 1
spsetprobcond(2,prob1to2)
val2to3:=[5,10,15,20] ! realzed value of rn(3) at nodes in stage 3
spsetrand(rn(3),val2to3)
prob2to3:=[0.2,0.3,0.4,0.1] ! conditional probabilities of visting
  ! stage 3 nodes from stage 2
spsetprobcond(3,prob2to3)
```

**Further information**

1. This procedure is useful when scenario tree structure is stage-wise symmetric, i.e., all the nodes in a stage have the same conditional probability distributions of visiting their children.
2. Tree can be generated by setting values of sprands in the tree and probabilities of visiting nodes or scenarios, followed by calling spgentree().

**Related topics**

```
spcreatetree, spsetprobscen, spsetprobcondatnode, spsetrandatnode,
spaddchildren, spgentree,
spprinttree
```
spaddchildren

Purpose
It is used for creating a scenario tree by progressively adding children to current node.

Synopsis
spaddchildren(t: integer, n: integer, C: integer)

Arguments
- t: stage number belonging to 1..T, where T is number of stages
- n: node number belonging to 1..N_t, where N_t is number of nodes in stage t
- C: number of Children nodes to be added to n^th node in the t^th stage

Example
Ex: create a 3-stage binary tree
spaddchildren(1,1,2)
spaddchildren(2,1,2)
spaddchildren(2,2,2)

Further information
1. All sprands must have been associated with stages before creating scenario tree.
2. The first call to spaddchildren() must begin with t=n=1 in order to facilitate proper initialization.
3. Tree can be generated by setting values of sprands in the tree and probabilities of visiting nodes or scenarios, followed by calling spgentree().

Related topics
- spsetprobscen, spsetprobcond, spsetprobcondatnode, spsetrand, spsetrandatnode,
- spcreatetree, spgentree,
- spprinttree
spsetprobscen

Purpose
It is used for setting the probabilities of scenarios in a scannerio tree.

Synopsis
spsetprobscen(s: integer, p: real)

Arguments
s scenario number belonging to 1..S, where S is number of scenarios
p probability of s\text{th} scenario

Further information
1. This procedure can be called only in a scenario based problem.
2. Tree can be generated by setting values of sprands in the tree and probabilities of visiting nodes or scenarios, followed by calling spgentree().

Related topics
spcreatetree, spsetprobcond, spsetprobcondatnode, spsetrand, spsetrandatnode, spaddchildren, spgentree, spprinttree
spsetprobcondatnode, spsetrandatnode

**Purpose**
It is used for setting conditional probabilities at all the nodes in a stage.

**Synopsis**

```
spsetprobcondatnode(t: integer, n: integer, p: real)
spsetrandatnode(rn: sprand, n: integer, v: real)
```

**Arguments**
- **t**: stage number belonging to 1..T, where T is number of stages
- **n**: node number belonging to 1..N_t, where N_t is number of nodes in stage t
- **p**: conditional probability of n^{th} node in the t^{th} scenario
- **rn**: stochastic random element
- **v**: realized value of rn at the n^{th} node in the t^{th} stage, where t is the stage number of rn

**Further information**
Tree can be generated by setting values of sprands in the tree and probabilities of visiting nodes or scenarios, followed by calling spgentree().

**Related topics**
- spcreatetree, spsetprobscen, spsetprobcond, spsetrand, spaddchildren, spgentree, spprinttree
**spgentree**

**Purpose**
It is used for finalizing scenario tree and verifying its structure and probabilities.

**Synopsis**
```
spgentree
```

**Further information**
This procedure should be called after setting values of sprands in the tree and probabilities of visiting nodes or scenarios.

**Related topics**
- `spsetdist`, `spgenexhtree`,
- `spcreateetree`, `spsetprobscen`, `spsetprobcond`, `spsetprobcondatnode`, `spsetrand`,
- `spsetrandatnode`,
- `spaddchildren`, `spprinttree`
maximize, minimize

**Purpose**

To optimize the stochastic problem.

**Synopsis**

maximize(obj: splinctr)
maximize(obj: splinctr, option: string)
maximize(obj: splinctr, t: integer, n: integer)
maximize(obj: splinctr, s: integer)
minimize(obj: splinctr)
minimize(obj: splinctr, option: string)
minimize(obj: splinctr, t: integer, n: integer)
minimize(obj: splinctr, s: integer)

**Arguments**

- **obj**  
  Objective function
- **option**  
  Options for choosing combination of algorithms and types of problems to be solved
  - XSP_REC  
    Solve Recourse problem
  - XSP_EV  
    Solve Expected Value problem
  - XSP_PI  
    Solve Perfect Information problem
  - XSP_OPT_LP_ONLY  
    Solve LP relaxation of the problem
  - XSP_PRIMAL  
    Solve problem using Primal algorithm
  - XSP_DUAL  
    Solve problem using Dual algorithm
  - XSP_BARRIER  
    Solve problem using Barrier algorithm
- **t**  
  stage number belonging to 1..T, where T is number of stages
- **n**  
  node number belonging to 1..N_t, where N_t is number of nodes in stage t
- **s**  
  scenario number belonging to 1..S, where S is number of scenarios

**Further information**

1. This procedure should be called after building scenario tree and stochastic constraints.
2. By default Recourse problem is solved.
3. Expected Value with Recourse can be solved using the option XSP_EV+XSP_REC.
4. Perfect Information with Recourse can be solved using the option XSP_PI+XSP_REC.
5. A problem instance in a particular scenario may be solved by passing the scenario number. The problem should be scenario based, i.e., xsp_scen_based should be true. User may want to prevent the overhead in this case by setting xsp_re_init to false.
6. A problem instance at a particular node in a scenario tree may be solved by passing the stage and the node number. The problem should be node based, i.e., xsp_scen_based should be false. User may want to prevent the overhead in this case by setting xsp_re_init to false.
7. Hidden constraints are not considered during optimization. If there are some terms in any constraint containing fixed variables, they are substracted off from the constraint’s right hand side.
8. Users should check the xprs_lpsstatus, or xprs_mipstatus and compare it to constants XSP_LP_-OPTIMAL, or XSP_MIP_-OPTIMAL resp to ensure that the current problem is optimal.

**Related topics**

getobjval, spgetobjconst,
getsol, getsolatnode, getsolinscen,
getdual, getdualatnode, getdualinscen,
getrcost, getrcostatnode, getrcostinscen,
getslack, getslackatnode, getslackinscen, 
speval, 
spfix, spunfix, spsethidden, 
spexportprob, spwriteprob
getobjval

Purpose
To obtain objective value of current stochastic problem after optimization.

Synopsis
getobjval: real

Return value
current objective value

Further information
1. This procedure should be called after optimization.
2. If objective function contains constants or fixed variables, their aggregated value is also included in the returned objective value.

Related topics
spgetobjconst,
getsol, getsolatnode, getsolinscen,
getdual, getdualatnode, getdualinscen,
getrcost, getrcostatnode, getrcostinscen,
getslack, getslackatnode, getslackinscen,
speval,
minimize, minimize
spgetobjconst

**Purpose**
To obtain objective value of constant term of the objective function.

**Synopsis**

```plaintext
spgetobjconst: real
```

**Return value**

- current constant term

**Further information**

1. This procedure should be called after optimization.
2. If objective function contains constants terms or terms containing fixed variables, then their total value is included in the objective's constant term.

**Related topics**

```
getobjval,
getsol, getsolatnode, getsolinscen,
getdual, getdualatnode, getdualinscen,
getrcost, getrcostatnode, getrcostinscen,
getsslack, getslackatnode, getslackinscen,
speval,
minimize, minimize
```
getsol, getsolatnode, getsolinscen

Purpose
To obtain solution of current stochastic problem after optimization.

Synopsis
getsol(x: spvar): real
getsolatnode(x: spvar, n: integer): real
getsolinscen(x: spvar, s: integer): real

Arguments
x stochastic variable
n node number belonging to 1..N_t, where x belongs to stage t, and N_t is number of nodes in stage t
s scenario number belonging to 1..S, where S is number of scenarios

Return value
current solution value

Further information
1. This procedure should be called after optimization.
2. x cannot be a fixed variable, and it must participate in current optimization problem.
3. If x belongs to first stage then solution can be obtained by calling getsol.
4. For Expected Value problems solution can be obtained by calling getsol.
5. For an instance of the problem in a scenario or at a node in a scenario tree, solution can be obtained by calling getsol.
6. For a node based problem, solution can be obtained by calling getsolatnode.
7. For a scenario based problem, solution can be obtained by calling getsolinscen.

Related topics
getobjval, spgetobjconst, speval, minimize, minimize,
getrcost, getrcostatnode, getrcostinscen, getdual, getdualatnode, getdualinscen, getslack, getslackatnode, getslackinscen
getrcost, getrcostatnode, getrcostinscen

Purpose
To obtain reduced cost of current stochastic problem after optimization.

Synopsis
getrcost(x: spvar): real
getrcostatnode(x: spvar, n: integer): real
getrcostinscen(x: spvar, s: integer): real

Arguments
x stochastic variable
n node number belonging to 1..N_t, where x belongs to stage t, and N_t is number of nodes in stage t
s scenario number belonging to 1..S, where S is number of scenarios

Return value
reduced cost

Further information
1. This procedure should be called after optimization.
2. x cannot be a fixed variable, and it must participate in current optimization problem.
3. If x belongs to first stage then solution can be obtained by calling getsol.
4. For Expected Value problems solution can be obtained by calling getsol.
5. For an instance of the problem in a scenario or at a node in a scenario tree, solution can be obtained by calling getsol.
6. For a node based problem, solution can be obtained by calling getsolatnode.
7. For a scenario based problem, solution can be obtained by calling getsolinscen.

Related topics
getobjval, spgetobjconst, speval, minimize, minimize,
getrcost, getrcostatnode, getrcostinscen, getdual, getdualatnode,
getdualinscen, getslack, getslackatnode, getslackinscen
getdual, getdualatnode, getdualinscen

Purpose
To obtain dual of current stochastic problem after optimization.

Synopsis
getdual(c: splinctr): real
getdualatnode(c: splinctr, n: integer): real
getdualinscen(c: splinctr, s: integer): real

Arguments
- c: stochastic linear constraint
- n: node number belonging to 1..Nt, where c belongs to stage t, and Nt is number of nodes in stage t
- s: scenario number belonging to 1..S, where S is number of scenarios

Return value
dual

Further information
1. This procedure should be called after optimization.
2. c cannot be a hidden contraint, and it must participate in current optimization problem.
3. If c belongs to first stage then dual can be obtained by calling getdual.
4. For Expected Value problem, dual can be obtained by calling getdual.
5. For an instance of the problem in a scenario or at a node in a scenario tree, dual can be obtained by calling getdual.
6. For a node based problem, dual can be obtained by calling getdualatnode.
7. For a scenario based problem, dual can be obtained by calling getdualinscen.

Related topics
getobjval, spgetobjconst, speval, minimize, minimize,
getcost, getcostatnode, getcostinscen, getsol, getsolatnode, getsolinscen,
getslack, getslackatnode, getslackinscen
getslack, getslackatnode, getslackinscen

Purpose
To obtain slack of current stochastic problem after optimization.

Synopsis
getslack(c: splinctr): real
getslackatnode(c: splinctr, n: integer): real
getslackinscen(c: splinctr, s: integer): real

Arguments
- c stochastic linear constraint
- n node number belonging to 1..N_t, where c belongs to stage t, and N_t is number of nodes in stage t
- s scenario number belonging to 1..S, where S is number of scenarios

Return value
- slack

Further information
1. This procedure should be called after optimization.
2. c cannot be a hidden constraint, and it must participate in current optimization problem.
3. If c belongs to first stage then dual can be obtained by calling getdual.
4. For Expected Value problem, dual can be obtained by calling getdual.
5. For an instance of the problem in a scenario or at a node in a scenario tree, dual can be obtained by calling getdual.
6. For a node based problem, dual can be obtained by calling getdualatnode.
7. For a scenario based problem, dual can be obtained by calling getdualinscen.

Related topics
- getobjval, spgetobjconst, speval, minimize, minimize,
- getcost, getcostatnode, getcostinscen, getsol, getsolatnode, getsolinscen,
- getdual, getdualatnode, getdualinscen
Purpose
To obtain the value of stochastic random element, expression, variable or linear constraint in a scenario tree.

Synopsis

\[
\begin{align*}
\text{speval}(\text{rn}: \text{sprand}): & \text{ real} \\
\text{speval}(\text{rn}: \text{sprand}, s: \text{ integer}): & \text{ real} \\
\text{speval}(\text{rn}: \text{sprand}, t: \text{ integer}, n: \text{ integer}): & \text{ real} \\
\text{speval}(\text{re}: \text{sprandexp}): & \text{ real} \\
\text{speval}(\text{re}: \text{sprandexp}, s: \text{ integer}): & \text{ real} \\
\text{speval}(\text{re}: \text{sprandexp}, t: \text{ integer}, n: \text{ integer}): & \text{ real} \\
\text{speval}(\text{x}: \text{spvar}): & \text{ real} \\
\text{speval}(\text{x}: \text{spvar}, s: \text{ integer}): & \text{ real} \\
\text{speval}(\text{x}: \text{spvar}, t: \text{ integer}, n: \text{ integer}): & \text{ real} \\
\text{speval}(\text{c}: \text{splinctr}): & \text{ real} \\
\text{speval}(\text{c}: \text{splinctr}, s: \text{ integer}): & \text{ real} \\
\text{speval}(\text{c}: \text{splinctr}, t: \text{ integer}, n: \text{ integer}): & \text{ real}
\end{align*}
\]

Arguments
\begin{align*}
x & \text{ stochastic variable} \\
c & \text{ stochastic linear constraint} \\
t & \text{ stage number } t \text{ belonging to } 1..T, \text{ where } T \text{ is the number of stages} \\
n & \text{ node number belonging to } 1..N_t, \text{ where } N_t \text{ is number of nodes in stage } t \\
s & \text{ scenario number belonging to } 1..S, \text{ where } S \text{ is number of scenarios}
\end{align*}

Return value
realized value of stochastic entity in scenario tree

Further information
1. The stage of \text{rn/re/x/c} cannot be greater than \text{t}.
2. If \text{x} is fixed, its fixed value is returned.
3. \text{speval(.)} returns fixed or expected value of stochastic entity depending on its.
4. \text{speval(.,t,n)} returns realized value of stochastic entity at the \text{n}\text{th} node in the \text{t}\text{th} stage.
5. \text{speval(.,s)} returns realized value of stochastic entity in the \text{s}\text{th} scenario.

Related topics
spfix

Purpose
To fix a stochastic variable to a value.

Synopsis
spfix(x: spvar, v: real)

Arguments
x stochastic variable
v The value to which x has to be fixed

Further information
1. Fixed variables do not participate in the columns of the matrix generated for optimization.
2. The terms consisting of fixed variables are substracted from the right hand side of a constraint consisting of these terms.
3. If all the variables in a constraint are fixed then the constraint is automatically hidden.

Related topics
spunfix, spsethidden, maximize, minimize
spunfix

Purpose
To unfix a stochastic variable that has been previously fixed.

Synopsis
spunfix(x: spvar)

Argument
x stochastic variable

Further information
1. This procedure is helpful in switching from say Expected Value problem with Recourse to the usual Recourse problem.
2. This procedure is useful for solving the multiple related problems iteratively.

Related topics
spfix, spsethidden, maximize, minimize
spsethidden

Purpose
To hide or unhide stochastic linear constraint.

Synopsis
spsethidden(c: splinctr, hide: boolean)

Arguments
- c: stochastic linear constraint
- hide: condition for hiding or unhiding c
  - true: hide c
  - false: unhide c

Further information
This procedure is useful for solving the multiple related problems iteratively.

Related topics
spfix, spunfix, maximize, minimize
spsettype

Purpose
Used for setting the type of a splinctr.

Synopsis
spsettype(c: splinctr, type: string)

Arguments
  c    stochastic linear constraint
  type type of c
       XSP_GLOBAL  set c as global constraint

Example

declarations
Z: spvar
Prob: splinctr
end-declarations
Z is_binary
spsetstage(Z,T) ! associating Z to last stage
Prob:=Z<=0.5
spsettype(Prob,XSP_GLOBAL) ! internally expanded as
! sum(s in 1..S) P(s).Z(s)<=0.5,
! i.e., Pr{event corresponding to Z=1}<=0.5

Further information
1. Global constraints are useful for modeling chance constraints.
2. Global constraints consist of variable terms across all scenarios, weighted by probabilities of the scenarios.
3. Global constraints are simply ignored in Expected Value problems.
4. Global constraints are simply ignored in an instance of the problem in a scenario or at a node in a scenario tree.
5. Global constraints cannot contain fixed variables.
6. Stochastic constraints can also be of type sos_1 and sos_2.
7. Stochastic variables can be of type is_binary, is_integer, or is_free.
spgetscencount

**Purpose**
To obtain total number of scenarios in a scenario tree.

**Synopsis**
```
spgetscencount: integer
```

**Return value**
$S$ number of scenarios

**Further information**
This function should be called after generating scenario tree.

**Related topics**
- `spgetnodecount`
- `spgetprobcond`, `spgetprobscen`, `spgetprobuncond`
- `spgetparent`, `spgetchildrencount`, `spgetchild`
- `spprinttree`
spgetnodecount

Purpose
To obtain total number of nodes in a stage in a scenario tree.

Synopsis
spgetnodecount(t:integer): integer

Argument
t stage number belonging to 1..T, where T is the number of stages

Return value
N(t) number of nodes in stage t

Further information
This function should be called after generating scenario tree.

Related topics
spgetscencount, spgetprobcond, spgetprobscen, spgetprobuncond, spgetparent, spgetchildrencount, spgetchild, spprinttree
spgetprobcond

Purpose
To obtain the conditional probability of visiting a node from its parent node.

Synopsis
spgetprobcond(t:integer, n: integer): real

Arguments
- t        stage number belonging to 1..T, where T is the number of stages
- n        node number belonging to 1..N_t, where N_t is number of nodes in stage t

Return value
p(t,n)    conditional probability of visiting n^{th} node in the t^{th} stage from its parent's node

Further information
This function should be called after generating scenario tree.

Related topics
spgetscencount, spgetnodecount,
spgetprobuncond, spgetprobscen,
spgetparent, spgetchildrencount, spgetchild,
spprinttree
**spgetprobuncond**

**Purpose**
To obtain the unconditional probability of visiting a node.

**Synopsis**

\[
\text{spgetprobuncond}(t: \text{integer}, n: \text{integer}): \text{real}
\]

**Arguments**

- \(t\)  
  stage number belonging to 1..\(T\), where \(T\) is the number of stages  
- \(n\)  
  node number belonging to 1..\(N_t\), where \(N_t\) is number of nodes in stage \(t\)

**Return value**

\(P(t, n)\)  
unconditional probability of visiting \(n^{th}\) node in the \(t^{th}\) stage

**Further information**

1. This function should be called after generating scenario tree.
2. The probability of a scenario is simply the unconditional probability of visiting the terminal node corresponding to that scenario.
3. The unconditional probability at a node is simply the product of conditional probabilities at nodes that belong to the path to that node.

**Related topics**

- spgetscencount, spgetnodecount,  
- spgetprobcond, spgetprobscen,  
- spgetparent, spgetchildrencount, spgetchild,  
- spprinttree
**spgetprobscen**

**Purpose**
To obtain the (unconditional) probability of a scenario.

**Synopsis**

\[
\text{spgetprobscen}(s: \text{integer}): \text{real}
\]

**Argument**

\[s\]
scenario number belonging to \(1..S\), where \(S\) is number of scenarios

**Return value**

\[P(s)\]
probability of occurrence of \(s^{th}\) scenario

**Further information**
This function should be called after generating scenario tree.

**Related topics**
- spgetscencount, spgetnodecount,
- spgetprobcond, spgetprobuncond,
- spgetparent, spgetchildrencount, spgetchild,
- spprinttree
spgetparent

Purpose
To get the node number of the parent of a node in a scenario tree.

Synopsis
spgetparent(t: integer, n: integer): integer

Arguments
- t: stage number belonging to 1..T, where T is the number of stages
- n: node number belonging to 1..N_t, where N_t is number of nodes in stage t

Return value
node number of parent of n_th node in the t_th stage

Further information
This function should be called after generating scenario tree.

Related topics
spgetscencount, spgetnodecount,
spgetprobcond, spgetprobuncond, spgetprobscen,
spgetchildrencount, spgetchild,
spprinttree
spgetchildrencount

**Purpose**
To get the number of children of a node in a scenario tree.

**Synopsis**
```
spgetchildrencount(t:integer, n: integer): integer
```

**Arguments**
- `t` stage number belonging to $1..T$, where $T$ is the number of stages
- `n` node number belonging to $1..N_t$, where $N_t$ is number of nodes in stage $t$

**Return value**
- $C$ number of children of the $n^{th}$ node in the $t^{th}$ stage

**Further information**
This function should be called after generating scenario tree.

**Related topics**
- spgetscencount, spgetnodecount,
- spgetprobcond, spgetprobuncond, spgetprobscen,
- spgetparent, spgetchild,
- spprinttree
spgetchild

Purpose
To get the node number of a child of a node in a scenario tree.

Synopsis
spgetchild(t:integer, n: integer, c: integer): integer

Arguments
- \( t \) stage number belonging to 1..T, where T is the number of stages
- \( n \) node number belonging to 1..N\( _t \), where N\( _t \) is the number of nodes in stage \( t \)
- \( c \) child number belonging to 1..C, where C is the number of children of \( n^{th} \) node in the \( t^{th} \) stage

Return value
the node number of the \( c^{th} \) child of \( n^{th} \) node in the \( t^{th} \) stage

Further information
This function should be called after generating scenario tree.

Related topics
spgetscencount, spgetnodecount,
spgetprobcond, spgetprobuncond, spgetprobscen,
spgetparent, spgetchildrencount,
spprinttree
**Purpose**
To delete a scenario or a set of scenarios from a scenario tree.

**Synopsis**
- `spdelscen(s: integer)`
- `spdelscen(scens: range)`
- `spdelscen(scens: set of integer)`

**Arguments**
- **s**: a single scenario to be deleted from current scenarios 1..S, where S is the number of scenarios.
- **scens**: subset of scenarios to be deleted from current scenarios 1..S, where S is the number of scenarios.

**Further information**
1. This procedure should be called after generating scenario tree.
2. The scenario tree structure gets updated after calling this procedure, implying the number of scenarios gets reduced, nodes are renumbered, and probabilities are renormalized.

**Related topics**
- `spaggregate`, `spgetscencount`, `spgetnodecount`,
- `spgetprobcond`, `spgetprobuncond`, `spgetprobscen`,
- `spgetparent`, `spgetchild`, `spgetchildrencount`,
- `spprinttree`
spaggregate

Purpose
To aggregate a set of scenarios from a scenario tree.

Synopsis
spaggregate(scens: array(range) of integer)
spaggregate(scens: range)
spaggregate(scens: set of integer)

Argument
scens  scenarios to be aggregated

Further information
1. This procedure should be called after generating scenario tree.
2. All or a subset of scenarios emerging from a common ancestral node may be aggregated.
3. The scenario tree structure gets updated after calling this procedure, implying the number of scenarios gets reduced, nodes are renumbered, and probabilities are added together.

Related topics
spdelscen, spgetscencount, spgetnodecount,
spgetprobcond, spgetprobuncond, spgetprobscen,
spgetparent, spgetchild, spgetchildrencount,
spprinttree
**spprinttree**

**Purpose**
To print the current scenario tree.

**Synopsis**
```
spprinttree
```

**Further information**
This procedure is helpful in visualizing the current tree in a text format for debugging purpose.

**Related topics**
- `spgetscencount`, `spgetnodecount`, `spgetprobcond`, `spgetprobuncond`, `spgetprobscen`, `spgetparent`, `spgetchildrencount`, `spgetchild`, `spsetdist`, `spgenexhtree`, `spcreatetree`, `spsetprobscen`, `spsetprobcond`, `spsetprobcondatnode`, `spsetrand`, `spsetrandatnode`, `spaddchildren`, `spgentree`
spexportprob

**Purpose**
To export the problem in the LP, SMPS, or SP format.

**Synopsis**
```plaintext
spexportprob(dir: integer, objfn: splincr, filetype: integer)
spexportprob(dir: integer, objfn: splincr, filetype: integer, filename: string)
```

**Arguments**
- `dir` direction for optimization
  - `XSP_MAX` maximization
  - `XSP_MIN` minimization
- `objfn` Objective function
- `filetype` type of file to be exported
  - `XSP_X_LP` lp
  - `XSP_X_SP` sp
  - `XSP_X_SMPS` smps
  - `XSP_X_SMPS_FREE` smps free format
- `filename` name of the file where the problem will be exported

**Further information**
1. If file name is not provided the name of the objective function is used for creating filename.
2. SMPS files consist of .tim, .sto, and .cor files.
3. SP file (.sp) is not readable by Xpress-SP. It is meant for debugging and verification purposes only.

**Related topics**
- `maximize`
- `minimize`
maximum, minimum

**Purpose**

To obtain the maximum or minimum.

**Synopsis**

```plaintext
maximum(r: real, rn: sprand): sprandexp
maximum(rn: sprand, r: real): sprandexp
maximum(r: real, re: sprandexp): sprandexp
maximum(re: sprandexp, r: real): sprandexp
maximum(rn: sprand, rn: sprand): sprandexp
maximum(rn: sprand, re: sprandexp): sprandexp
maximum(re: sprandexp, rn: sprand): sprandexp
maximum(re1: sprandexp, re2: sprandexp): sprandexp
minimum(r: real, rn: sprand): sprandexp
minimum(rn: sprand, r: real): sprandexp
minimum(re: sprandexp, r: real): sprandexp
minimum(rn: sprand, re: sprandexp): sprandexp
minimum(rn: sprand, re: sprandexp): sprandexp
minimum(re: sprandexp, rn: sprand): sprandexp
minimum(re1: sprandexp, re2: sprandexp): sprandexp
```

**Related topics**

`abs, ceil, exp, floor, ln, round, spif`
abs, ceil, floor

Purpose
To obtain absolute, ceil, floor, round, exponent, or natural log of stochastic entity.

Synopsis
abs(rn: sprand): sprandexp  
ceil(rn: sprand): sprandexp  
floor(rn: sprand): sprandexp  
round(rn: sprand): sprandexp  
exp(rn: sprand): sprandexp  
ln(rn: sprand): sprandexp  
abs(re: sprandexp): sprandexp  
ceil(re: sprandexp): sprandexp  
floor(re: sprandexp): sprandexp  
round(re: sprandexp): sprandexp  
exp(re: sprandexp): sprandexp  
ln(re: sprandexp): sprandexp

Related topics
maximum, minimum, spif
spif

Purpose
To obtain a certain value depending on the condition.

Synopsis

\begin{verbatim}
spif(cn: sprandexp, r1: real, r2: real): sprandexp
spif(cn: sprandexp, r: real, rn: sprand): sprandexp
spif(cn: sprandexp, r: real, re: sprandexp): sprandexp
spif(cn: sprandexp, rn: sprand, r: real): sprandexp
spif(cn: sprandexp, rn1: sprand, rn2: sprand): sprandexp
spif(cn: sprandexp, rn: sprand, re: sprandexp): sprandexp
spif(cn: sprandexp, re: sprandexp, r: real): sprandexp
spif(cn: sprandexp, re: sprandexp, rn: sprand): sprandexp
spif(cn: sprandexp, re1: sprandexp, re2: sprandexp): sprandexp
\end{verbatim}

Arguments

1st argument  stochastic boolean condition
2nd argument  if cn is evaluated to true, this value is returned as a random expression
3rd argument  otherwise this value is returned as a random expression

Example

assuming sr1, sr2, and sr3 are of type sprand, and sre is of type sprandexp, following is a valid call

\begin{verbatim}
sre:=spif(sr1<=3 and sr1>=sr2^sr3, 1, maximum(sr2,ceil(sr1)))
\end{verbatim}

Further information

1. Relational operators:  \leq, \geq, =, or logical operators: and, or, may be combined with types: integers, reals, sprands, sprandexps to create boolean stochastic random expressions.

2. spif() may also be used for modeling conditional stochastic functions.

Related topics

maximum, minimum, abs, ceil, exp, floor, ln, round
**spsetname**

**Purpose**
Used for setting the names of spvar, splinctr, or sprand.

**Synopsis**

```
spsetname(rn: sprand, name: string)
spsetname(x: spvar, name: string)
spsetname(c: splinctr, name: string)
```

**Arguments**

- **rn**: stochastic random element
- **x**: stochastic variable
- **c**: stochastic linear constraint
- **name**: name of rn/x/c

**Example**

```
forall(t in 1..T) spsetname(x(t),"var["+t+"]")
```

**Further information**

Names are automatically set. Users may override this by setting names themselves.
Chapter 8
Parameters

xsp_cond Prob  control for conditional probabilities  p. 130
xsp_create_tree control for creating scenario tree  p. 130
xsp_disp_warnings enable or disable warning messages  p. 128
xsp_implicit_stage associate stochastic entities with stages automatically  p. 128
xsp_ive_enable control for visualization in Xpress-IVE  p. 129
xsp_loadnames control for loading names  p. 129
xsp_re_init control for reinitialization  p. 128
xsp_scale_probs control for re-scaling probabilities  p. 129
xsp_scen_based Produce a scenario based or a node based problem  p. 127
xsp_tolprob_lb tolerance on lower bound of probabilities  p. 130
xsp_tolprob_totlb tolerance on lower bound of probabilities  p. 131
xsp_tolprob_totub tolerance on upper bound of probabilities  p. 131
xsp_tolprob_ub tolerance on upper bound of probabilities  p. 130
xsp_verbose enable or disable optimizer messages  p. 128

xsp_scen_based

Description This parameter is used for expanding the model internally based on scenarios or on nodes of the scenario tree.

Type Boolean, read/write

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>expand problem based on scenarios</td>
</tr>
<tr>
<td>false</td>
<td>expand problem based on nodes</td>
</tr>
</tbody>
</table>

Default value false

Affects routines maximize, minimize, spdelscen, spaggregate, XSP_COND_PROB
**xsp_implicit_stage**

Description: This parameter can be used for associating stochastic random elements, and variables with stages automatically.

Type: Boolean, read/write

Values:
- **true**: implicitly associate
- **false**: do not associate

Default value: false

Notes: User may override the stages set automatically by explicitly setting the stages using the function `spsetstage`. If y is spvar, and x is an array(1..10,Stages) of spvar then by setting XSP_implicit_STAGE to **true**, y's stage will be set to 1, and x(i,t)'s stage will be set to t.

**xspVerbose**

Description: This parameter is used for controlling optimizer messages.

Type: Boolean, read/write

Values:
- **true**: show messages
- **false**: do not show messages

Default value: true

**xspDispWarnings**

Description: This parameter is used for controlling Xpress-SP warning messages.

Type: Boolean, read/write

Values:
- **true**: show messages
- **false**: do not show messages

Default value: true

**xspReInit**

Description: This parameter controls whether change in structure of the problem from the previous run be accounted in the current run.

Type: Boolean, read/write

Values:
- **true**: reinitialize
- **false**: do not reinitialize

Default value: true
### Notes

If the size and structure of the underlying problem doesn't change from the previous run of the optimization, and only data changes, e.g., if no new variables or constraints are added or deleted then it is more efficient to prevent reinitialization. E.g., if user is solving problem instance in different scenarios, then this parameter may be set to false, since the problem would essentially remain same except the realized value of stochastic coefficients.

---

#### xsp_ive_enable

<table>
<thead>
<tr>
<th>Description</th>
<th>This parameter controls whether IVE be permitted to access model’s stochastic entities.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean, read/write</td>
</tr>
</tbody>
</table>
| Values      | true: allow access  
false: do not allow access |
| Default value | true                                      |
| Notes       | If user is optimizing repeatedly then it may be more efficient to disallow access to IVE. |

---

#### xsp_loadnames

<table>
<thead>
<tr>
<th>Description</th>
<th>This parameter controls whether names of stochastic entities be used in the underlying matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean, read/write</td>
</tr>
</tbody>
</table>
| Values      | true: load names  
false: do not load names |
| Default value | true                                       |
| Notes       | If XSP_LOADNAMES=false then the names of columns and rows in the matrix are decided by the Xpress-optimizer, and the names of variables and constraints in the exported problems are decided by Xpress-SP. |

---

#### xsp_scale_probs

<table>
<thead>
<tr>
<th>Description</th>
<th>This parameter controls whether conditional, unconditional and scenario probabilities be re-scaled.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean, read/write</td>
</tr>
</tbody>
</table>
| Values      | true: re-scale  
false: do not re-scale |
| Default value | true                                           |
| Notes       | Typically probabilities viz conditional probabilities of visiting children nodes should sum to 1. If due to numerical discrepancy they do not sum to 1, then by setting XSP_SCALE_PROBS=1 they can be normalized. |
**xsp_cond_prob**

**Description**
This parameter controls whether product of conditional probabilities in a scenario should equal the scenario probability.

**Type**
Boolean, read/write

**Values**
- true is equal
- false is not equal

**Default value**
true

**Set by routines**
XSP_SCEN_BASED

**Notes**
In a node based problem this parameter is always true.

**xsp_create_tree**

**Description**
This parameter controls whether the random elements and conditional probabilities be set in the scenario tree.

**Type**
Boolean, read/write

**Values**
- true create full tree
- false only create tree structures

**Default value**
true

**Notes**
If stochastic random elements have discretized distributions and creating exhaustive scenario is unnecessary, e.g., if the user doesn’t wish to optimize the problem but just export it in SMPS format, then this parameter may be set to false.

**xsp_tolprob_lb**

**Description**
This parameter controls lower bound on probabilities in probability distributions, conditional probabilities, unconditional probabilities, and scenario probabilities.

**Type**
Double, read/write

**Default value**
1e-6

**Notes**
probability ≥ XSP_TOLPROB_LB

**xsp_tolprob_ub**

**Description**
This parameter controls upper bound on probabilities in probability distributions, conditional probabilities, unconditional probabilities, and scenario probabilities.

**Type**
Double, read/write

**Default value**
1e-6

**Notes**
probability ≤ 1+ XSP_TOLPROB UB
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Default value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsp_tolprob_totlb</td>
<td>This parameter controls lower bound on probabilities in probability distributions, conditional probabilities, unconditional probabilities, and scenario probabilities.</td>
<td>Double, read/write</td>
<td>1e-6</td>
<td>total probability ≥ 1 - XSP_TOLPROB_TOTLB</td>
</tr>
<tr>
<td>xsp_tolprob_totub</td>
<td>This parameter controls upper bound on probabilities that should ideally sum to 1 (total conditional probability, total unconditional probability, total scenario probability).</td>
<td>Double, read/write</td>
<td>1e-6</td>
<td>total probability ≤ 1 + XSP_TOLPROB_TOTUB</td>
</tr>
</tbody>
</table>
Appendix
Appendix A
Solution techniques

Various specialized algorithms can be used to solve stochastic problems. These algorithms can further be categorized, but in this section, we outline merely the generalized form of these algorithms.

• 2-stage:
  – **LP:** Stochastic linear programs are typically solved using the **L-shaped method.** This method falls in to the category of **outer linearization** methods where cuts (feasibility and optimality) are generated to solve the problem faster. The most prevalent versions of L-shaped method are single cut and multi-cut. LPs are also solved using the **inner linearization** methods, which essentially involved column generation by using the Dantzig-Wolfe decomposition method. Other methods such as basis factorization are quite intensive computationally.
  – **IP:** For integer problems, the **Integer L-shaped method** is used. If the 2-stage problem consists of binary first stage variables, then special kinds of cuts can be generated in order to obtain the solutions faster. Special techniques for MIP’s can also be applied by decomposing the second stage variables in a 2-stage problem into discrete and continuous parts.

• Multi-stage:
  – **Nested Decomposition methods** are available for solving the multi-stage LP. The algorithm involves column generation from the previous sub-problems and using them in the current period master problem. The LP are solved by **Nested L-shaped methods.**
Appendix B
Example problem data

B.1 Data for the airlift scheduling model

<table>
<thead>
<tr>
<th>airlift1.dat:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: ( [(1,1) \ 24 \ (1,2) \ 14 \ (2,1) \ 49 \ (2,2) \ 29] )</td>
</tr>
<tr>
<td>Aswitch: ( [(1,1,2) \ 19 \ (1,2,1) \ 29 \ (2,1,2) \ 18 \ (2,2,1) \ 26] )</td>
</tr>
<tr>
<td>F: ( [7200 \ 7200] )</td>
</tr>
<tr>
<td>b: ( [(1,1) \ 50 \ (1,2) \ 75 \ (2,1) \ 60 \ (2,2) \ 40] )</td>
</tr>
<tr>
<td>Cost: ( [(1,1) \ 7200 \ (1,2) \ 6000 \ (2,1) \ 7200 \ (2,2) \ 4000] )</td>
</tr>
<tr>
<td>CostSwitch: ( [(1,1,2) \ 7000 \ (1,2,1) \ 8200 \ (2,1,2) \ 5500 \ (2,2,1) \ 8700] )</td>
</tr>
<tr>
<td>cplus: ( [500 \ 250] )</td>
</tr>
<tr>
<td>cminus: ( [0 \ 0] )</td>
</tr>
<tr>
<td>DValues: ( [] )</td>
</tr>
<tr>
<td>(1,1): 927.758357</td>
</tr>
<tr>
<td>(1,2): 982.516248</td>
</tr>
<tr>
<td>(1,3): 961.404897</td>
</tr>
<tr>
<td>(1,4): 922.915716</td>
</tr>
<tr>
<td>(1,5): 986.342969</td>
</tr>
<tr>
<td>(1,6): 999.134104</td>
</tr>
<tr>
<td>(1,7): 970.324386</td>
</tr>
<tr>
<td>(1,8): 949.613106</td>
</tr>
<tr>
<td>(1,9): 991.773703</td>
</tr>
<tr>
<td>(1,10): 979.491162</td>
</tr>
<tr>
<td>(1,11): 979.679661</td>
</tr>
<tr>
<td>(1,12): 964.052640</td>
</tr>
<tr>
<td>(1,13): 957.691777</td>
</tr>
<tr>
<td>(1,14): 930.372603</td>
</tr>
<tr>
<td>(1,15): 933.799027</td>
</tr>
<tr>
<td>(1,16): 995.204085</td>
</tr>
<tr>
<td>(1,17): 957.344884</td>
</tr>
<tr>
<td>(1,18): 923.483118</td>
</tr>
<tr>
<td>(1,19): 959.026809</td>
</tr>
<tr>
<td>(1,20): 946.706588</td>
</tr>
<tr>
<td>(1,21): 991.897924</td>
</tr>
<tr>
<td>(1,22): 956.965721</td>
</tr>
<tr>
<td>(1,23): 981.616042</td>
</tr>
<tr>
<td>(1,24): 957.688286</td>
</tr>
<tr>
<td>(1,25): 1000.035618</td>
</tr>
<tr>
<td>(2,1): 1433.626750</td>
</tr>
<tr>
<td>(2,2): 1149.727635</td>
</tr>
<tr>
<td>(2,3): 1492.817415</td>
</tr>
<tr>
<td>(2,4): 1250.557154</td>
</tr>
<tr>
<td>(2,5): 1353.935057</td>
</tr>
<tr>
<td>(2,6): 1226.355338</td>
</tr>
<tr>
<td>(2,7): 1378.148830</td>
</tr>
<tr>
<td>(2,8): 1200.624255</td>
</tr>
<tr>
<td>(2,9): 1045.317699</td>
</tr>
<tr>
<td>(2,10): 899.933450</td>
</tr>
<tr>
<td>(2,11): 1439.677972</td>
</tr>
<tr>
<td>(2,12): 1170.804834</td>
</tr>
<tr>
<td>(2,13): 1474.836349</td>
</tr>
<tr>
<td>(2,14): 1572.684651</td>
</tr>
<tr>
<td>(2,15): 1207.662826</td>
</tr>
</tbody>
</table>
B.2 Data for the forest planning problem

forest.dat:

yield:
[0 0 16 107 275 298 306]
v: [320.3417 356.1874 398.4370 448.2349 506.9294 564.9294 587.9294 595.9294]
alpha: 0.9
beta: 1.1
delta: 0.905
gamma: 50
ProbDiscret:
[1.0000 0.0000 0.0000 0.4616 0.5384 0.0000 0.1847 0.2769 0.5384]
ValuesDiscret:
[0.06258 0.00000 0.00000 0.08612 0.04240 0.00000 0.10499 0.07354 0.04240]
s1: [241 125 1404 2004 9768 16385 2815 61995]

B.3 Asset liability management model

Table B.1: Optimal objective values for different values of ψ.

<table>
<thead>
<tr>
<th>ψ</th>
<th>Zₚ</th>
<th>Zₑv</th>
<th>Zₚₑv</th>
<th>Zₚ - Zₑv</th>
<th>Zₚₑv - Zₚ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>-0.202383</td>
<td>-0.622399</td>
<td>-1.08452</td>
<td>0.420016</td>
<td>0.882137</td>
</tr>
<tr>
<td>1.25</td>
<td>-0.0470575</td>
<td>-0.484575</td>
<td>-0.965948</td>
<td>0.4375175</td>
<td>0.9188905</td>
</tr>
<tr>
<td>1.3</td>
<td>0.108268</td>
<td>-0.34675</td>
<td>-0.847378</td>
<td>0.455018</td>
<td>0.955646</td>
</tr>
<tr>
<td>1.35</td>
<td>0.263594</td>
<td>-0.208925</td>
<td>-0.728808</td>
<td>0.472519</td>
<td>0.992402</td>
</tr>
<tr>
<td>1.4</td>
<td>0.418919</td>
<td>-0.0711</td>
<td>-0.610238</td>
<td>0.490019</td>
<td>1.029157</td>
</tr>
</tbody>
</table>

Table B.2: Investment decisions (scaled values) for different values of ψ.

<table>
<thead>
<tr>
<th>stocks</th>
<th>ψ</th>
<th>1.2</th>
<th>1.25</th>
<th>1.3</th>
<th>1.35</th>
<th>1.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.127183</td>
<td>0.132482</td>
<td>0.137781</td>
<td>0.143081</td>
<td>0.14838</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.128142</td>
<td>0.133481</td>
<td>0.13882</td>
<td>0.144159</td>
<td>0.149499</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.622716</td>
<td>0.648663</td>
<td>0.674609</td>
<td>0.700556</td>
<td>0.726502</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.007406</td>
<td>0.007715</td>
<td>0.008023</td>
<td>0.008332</td>
<td>0.008641</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0.002992</td>
<td>0.003116</td>
<td>0.003241</td>
<td>0.003366</td>
<td>0.00349</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.003805</td>
<td>0.003964</td>
<td>0.004123</td>
<td>0.004281</td>
<td>0.00444</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.031253</td>
<td>0.032555</td>
<td>0.033857</td>
<td>0.03516</td>
<td>0.036462</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.000323</td>
<td>0.000336</td>
<td>0.000349</td>
<td>0.000363</td>
<td>0.000376</td>
<td></td>
</tr>
</tbody>
</table>
### Table B.3: Optimal contribution rates after deletion of scenarios

<table>
<thead>
<tr>
<th>m</th>
<th>No. of scenarios after deletion</th>
<th>Optimal solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>209</td>
<td>0.0560891</td>
</tr>
<tr>
<td>1.5</td>
<td>1614</td>
<td>0.0849195</td>
</tr>
<tr>
<td>2</td>
<td>3357</td>
<td>0.0947222</td>
</tr>
<tr>
<td>2.5</td>
<td>4524</td>
<td>0.0995853</td>
</tr>
<tr>
<td>3</td>
<td>4899</td>
<td>0.107498</td>
</tr>
<tr>
<td>3.5</td>
<td>4975</td>
<td>0.107498</td>
</tr>
<tr>
<td>4</td>
<td>4995</td>
<td>0.108268</td>
</tr>
</tbody>
</table>

#### B.3.1 Code for deleting the ‘extreme’ scenarios

The scenarios falling outside the intervals $[\mu_n - m \sigma_n, \mu_n + m \sigma_n]$ can be deleted from the scenario tree by adding the following code into the initial Mosel program after the line Spgentree and before the model definition.

```mosel
Declarations
  ! Statistical characteristics of the scenarios set
  means, stdev: array(Stocks) of real
  Eset: set of integer ! Set of "extreme" scenarios
  m: real ! Length of the interval
end-declarations

initializations from 'means.dat'
  means ! Mean returns of each of 12 stocks
end-initializations

initializations from 'stdev.dat'
  stdev ! Standard deviations of the returns
end-initializations

Eset:={} ! Initialize the set of extreme scenarios as an empty set
m:=2

forall(s in Scen, p in Stocks)
  if abs(Values(s,p)-means(p)) > m * stdev(p) then
    Eset:=Eset+{s} ! Update the set of extreme scenarios
    break
  end-if
spdelscen(Eset) ! Delete the set of extreme scenarios
```

Changing the value of $m$ allows one to delete different numbers of scenarios. Instead of deleting the extreme scenarios, one can aggregate them using `spaggregate(Eset)` instead of `spdelscen(Eset)`. In our case, the solutions of the problem with aggregated extreme scenarios are identical to the solutions with deleted extreme scenarios presented above.

**ALM.dat**:

```
Initial_Asset: 851826105
Initial_payment: 22170020
Total_wages: 211097880
```

#### B.3.2 Description of other data files

- **data.dat**: matrix with 500 rows and 12 columns, where each column represents 500 realizations of the return of a given stock
- **data_L.dat**: array of 500 elements representing the realizations of the amount of liabilities
- **means.dat**: array of 12 elements representing the mean return of each stock
- **stdev.dat**: array of 12 elements representing standard deviations of the returns
Appendix C

Xpress-SP error codes

E-105  *cannot aggregate these scenarios*
  Scenarios should have a common ancestral node

E-108  *cannot load the matrix*

E-109  *expected value problem is infeasible*
  Expected Value problem with Recourse cannot be solved

E-110  *illegal division by zero*

E-111  *invalid scenario tree*

E-116  *perfect information problem infeasible*

E-117  *probabilities for all the branches not set*
  Conditional probabilities at certain nodes are not set

E-118  *probabilities not within tolerances*

E-120  *scenarios not generated yet*

E-123  *sprands in the first stage should take one value with probability 1.0*
  First stage stochastic random elements are deterministic

E-133  *scenario tree does not exist*

E-134  *wrong scenario number entered*

E-135  *sprand does not exist*

E-136  *sprandexp does not exist*

E-137  *spvar does not exist*

E-138  *splinctr does not exist*

E-139  *node does not exist*

E-140  *global constraints can occur only in a recourse problem*

E-141  *cannot set the bound of spvar through (unnamed) splinctr*
  Unnamed constraints are assumed be bounds. A bound should be of the form $x \leq a$ or $x \geq a$, where $x$ is spvar and $a$ is real, sprand or sprandexp

E-143  *col / row not found in the matrix*

E-145  *cannot build scenarios*

E-146  *cannot evaluate at earlier stages*
  The value of the stochastic entity is realized at a later stage
E-148  **Problem type and entity attribute conflict**
Entity must belong to 1st stage or an expected value prob or a problem at a node / in a scenario or be a global, and not fixed / hidden. A reference to scenario number or a node number is required

E-149  **Problem type and entity attribute conflict**
Entity entity belongs to an expected value problem or a problem at a node / in a scenario, or is fixed / hidden / global. A reference to scenario number or a node number should not be provided

E-151  **Cannot have fixed variables in sos type constraints**

E-152  **Sos should be unconstrained**

E-153  **Invalid stage of stochastic bound for spvar**
The stage number of stochastic bound should be ≤ that of stochastic variable

E-155  **Constraints or bounds pertaining to fixed variables are not satisfied**
One or more constraints containing only fixed variables are being violated, given the values of fixed variables

W-100  **Duplicate entry found**

W-101  **Fixing variables and hiding constraints**

W-102  **Hiding constraint, since all variables are fixed**
If all variables in a constraint are fixed then the constraint is automatically hidden

W-103  **Ignoring invalid splinctr**

W-104  **Scenarios already generated; generating again from scratch**

W-105  **Setting negative lower bound of a non-free spvar**

W-106  **Setting xsp_cond_prob to true because xsp_scen_based is false**

W-107  **Sprands do not assume any realized values**

W-108  **Sprands in the first stage cannot be random**

W-109  **Stage already set**

W-110  **Sense of constraint is unchanged**

W-111  **Lower bound set to -inf**

W-112  **Changing bounds**

W-113  **Stage not set or invalid stage**

W-114  **Ignoring unused spvar**

W-107  **If a variable does not participate in any constraint, it is automatically rejected**

W-115  **Implicitly stages not set**

W-116  **Special ordered sets used only in recourse, expected value or perfect information problems**
Bibliography


Index

A
abs, 14, 124

C
ceil, 14, 124
chance constraint, 34

E
E-105, 137
E-108, 137
E-109, 137
E-110, 137
E-111, 137
E-116, 137
E-117, 137
E-118, 137
E-120, 137
E-123, 137
E-133, 137
E-134, 137
E-135, 137
E-136, 137
E-137, 137
E-138, 137
E-139, 137
E-140, 137
E-141, 137
E-143, 137
E-145, 137
E-146, 137
E-148, 138
E-149, 138
E-151, 138
E-152, 138
E-153, 138
E-155, 138
EV, see expected value
exp, 14
expected value problem, 10

F
floor, 14, 124

G
getdual, 14, 104
getdualatnode, 14, 104
getdualinscen, 14, 104
getobjval, 14, 100
getcost, 14, 103
getcostatnode, 14, 103
getcostinscen, 14, 103
getslack, 14, 105
getslackatnode, 14, 105
getslackinscen, 14, 105
getsol, 14, 102
getsolatnode, 14, 102
getsolinscen, 14, 102
global constraint, 34

I
inner linearization, 133
IVE, 35

L
L-shaped method, 133
ln, 14

M
maximize, 14, 98
maximum, 14, 123
minimize, 14, 98
minimum, 14, 123

N
nested decomposition, 133

O
outer linearization, 133

P
perfect information problem, 11
PI, see perfect information

R
recourse problem, 9
round, 14

S
scenario tree, 6
SP, see Stochastic Programming
spaddchildren, 14, 94
spaggregate, 14, 120
spcreatetree, 14, 92
spdelscen, 14, 119
speval, 14, 106
spexportprob, 14, 122
spfix, 14, 107
spgenexh, 91
spgenexhtree, 14
spgentree, 14, 97
spgetchild, 14, 118
spgetchildrencount, 14, 117
spgetnodecount, 14, 112
spgetobjconst, 101
spgetparent, 14, 116
spgetprobcond, 14, 113
spgetprobscen, 14, 115
spgetprobuncond, 14, 114
spgetsencount, 14, 111
spgetxprsintattrib, 14

140 Xpress-SP Reference Manual
### Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>spgetxprsrealattrib</td>
<td>14</td>
</tr>
<tr>
<td>spgetxprsstrattrib</td>
<td>14</td>
</tr>
<tr>
<td>spif</td>
<td>14, 125</td>
</tr>
<tr>
<td>spprinttree</td>
<td>14, 121</td>
</tr>
<tr>
<td>spsetdist</td>
<td>14, 89</td>
</tr>
<tr>
<td>spsethidden</td>
<td>14, 109</td>
</tr>
<tr>
<td>spsetname</td>
<td>14, 126</td>
</tr>
<tr>
<td>spsetprobcond</td>
<td>14, 93</td>
</tr>
<tr>
<td>spsetprobcondatnode</td>
<td>14, 96</td>
</tr>
<tr>
<td>spsetprobscen</td>
<td>14, 95</td>
</tr>
<tr>
<td>spsetrand</td>
<td>14, 93</td>
</tr>
<tr>
<td>spsetrandatnode</td>
<td>14, 96</td>
</tr>
<tr>
<td>spsetstage</td>
<td>14, 88</td>
</tr>
<tr>
<td>spsetstages</td>
<td>14, 87</td>
</tr>
<tr>
<td>spsettype</td>
<td>14, 110</td>
</tr>
<tr>
<td>spunfix</td>
<td>14, 108</td>
</tr>
<tr>
<td>spwritetext</td>
<td>14</td>
</tr>
</tbody>
</table>

### Stochastic Programming

**W**

- W-100, 138
- W-101, 138
- W-102, 138
- W-103, 138
- W-104, 138
- W-105, 138
- W-106, 138
- W-107, 138
- W-108, 138
- W-109, 138
- W-110, 138
- W-111, 138
- W-112, 138
- W-113, 138
- W-114, 138
- W-115, 138
- W-116, 138

### X

- XSP_BARRIER, 14, 31
- xsp_cblog, 14
- XSP_COND_PROB, 34
- xsp_cond_prob, 14, 130
- XSP_CREATE_TREE, 34
- xsp_create_tree, 14, 130
- XSP_DISP_WARNINGS, 34
- xsp_disp_warnings, 14, 128
- XSP_DUAL, 14, 31
- XSP_EV, 14, 31
- xsp_ftable, 14
- XSP_GLOBAL, 14
- XSP_IMPLICIT_STAGE, 34
- xsp_implicit_stage, 14, 128
- XSP_IVE_ENABLE, 33
- xsp_ive_enable, 14, 129
- XSP_LOADNAMES, 34
- xsp_loadnames, 14, 129
- XSP_LP_CUTOFF, 14, 32
- xsp_LP_CUTOFF_IN_DUAL, 14, 32
- XSP_LP_CUTOFF_IN_DUAL, 14, 32
- XSP_LP_INFEAS, 14, 32
- XSP_LP_OPTIMAL, 14, 32
- XSP_LP_UNBOUNDED, 14, 32
- XSP_LP_UNFINISHED, 14, 32
- XSP_MIN, 14
- XSP_MIP_INFES, 14, 32
- XSP_MIP_LP_NOT_OPTIMAL, 14, 32
- XSP_MIP_LP_OPTIMAL, 14, 32
- XSP_MIP_NO_SOL_FOUND, 14, 32
- XSP_MIP_NOT_LOADED, 14, 32
- XSP_MIP_OPTIMAL, 14, 32
- XSP_MIP_SOLUTION, 14, 32
- XSP_OPT_LP_ONLY, 14, 31
- XSP_PI, 14, 31
- XSP_PRIMAL, 14, 31
- XSP_RE_INIT, 34
- xsp_re_init, 14, 128
- XSP_REC, 14, 31
- XSP_SCALE_PROBS, 34
- xsp_scale_probs, 14, 129
- XSP_SCEN_BASED, 34
- xsp_scen_based, 14, 127
- XSP_SMPS_FREE, 14
- XSP_STAGE_ZERO, 14
- XSP_TOLPROB_LB, 34
- xsp_tolprob_l, 14, 130
- XSP_TOLPROB_TOTLB, 34
- xsp_tolprob_totl, 14, 131
- XSP_TOLPROB_TOTUB, 34
- xsp_tolprob_totu, 14, 131
- XSP_TOLPROB UB, 34
- xsp_tolprob_ub, 14, 130
- XSP_VERBOSE, 34
- xsp_verbose, 14, 128
- XSP_X_LP, 14
- XSP_X_SMPS, 14
- XSP_X_SP, 14
- xsp_xprs_prob, 14