Evaluation of scenario-generation methods for stochastic programming

Michal Kaut
michal.kaut@iot.ntnu.no[*]

Stein W. Wallace
stein.w.wallace@himolde.no[†]

May 7, 2003

Abstract

In this paper, we discuss the evaluation of quality/suitability of scenario-generation methods for a given stochastic programming model. We formulate minimal requirements that should be imposed on a scenario-generation method before it can be used for solving the stochastic programming model. We also show how the requirements can be tested.

The procedure of testing a scenario-generation method is illustrated on a case from portfolio management. In addition, we provide a short overview of the most common scenario-generation methods.

Keywords: stochastic programming, scenario tree, scenario generation

1 Introduction

In recent years, stochastic programming has gained an increasing popularity within the mathematical programming community. Present computing power allows users to add stochasticity to models that had been difficult to solve as deterministic models only a few years ago. In this context, a stochastic programming model can be viewed as a mathematical programming model with uncertainty about the values of some of the parameters. Instead of single values, these parameters are then described by distributions (in a single-period case), or by stochastic processes (in a multi-period case). A single-period stochastic programming model can thus be formulated ([20]) as:

$$\begin{aligned} \text{“min”}\ & g_0(\boldsymbol{x},\tilde{\boldsymbol{\xi}}) \\ \text{s.t.}\ & g_i(\boldsymbol{x},\tilde{\boldsymbol{\xi}}) \le 0,\ i = 1,\dots,m \\ & x \in \mathbf{X} \subset \mathbb{R}^n \ , \end{aligned} \tag{1}$$

where $\tilde{\boldsymbol{\xi}}$ is a random vector, whose distribution must be independent of the decision vector $\boldsymbol{x}$. Note that the formulation is far from complete—we still need to specify the meanings of “min” and the constraints.

Except for some trivial cases, (1) can not be solved with continuous distributions—most solution methods need discrete distributions. In addition, the cardinality of the support of the discrete distributions is limited by the available computing power, together with a complexity of the decision

[*]Norwegian University of Science and Technology, N-7491 Trondheim, Norway
[†]Molde University College, Postboks 2110, N-6402 Molde, Norway

model. Hence, in most practical applications, the distributions of the stochastic parameters have to be approximated by discrete distributions with a limited number of outcomes. The discretization is usually called a *scenario tree* or an event tree – see Figure 1 for an example.

Hence, we solve only an approximation of (1), with the quality of the approximation directly linked to a quality of the scenario tree: *garbage in, garbage out* holds here as anywhere else. Surprisingly, there has been little focus on measuring the quality of scenario trees. In this paper, we thus ask the question of what is a good scenario-generation method for a given stochastic programming model. The link to the decision model is very important, we do not believe there is a scenario-generation method that would be best for all possible models, even if these models were subject to the same random phenomena.

When comparing scenario-generation methods, we focus on practical performance, not on the theoretical properties: it may be comforting to know that a certain method approximates the distribution perfectly when the number of outcomes goes to infinity, yet it does not mean that the method is good for generating a tree with just a few scenarios. Indeed, some of the methods mentioned in Section 2 do not guarantee convergence to the true distribution, but perform very well in real-life problems. For more information on the theoretical properties, see for example [8].

Because of the variety of both scenario-generation methods and decision models, we do not provide a guideline of the type "for this model use that method". Instead, we formulate two important properties that a scenario-generation method should satisfied in order to be usable for a given model. We also show how to test the properties. The user can thus test several scenario-generation methods, and choose the one that is best suitable for the given decision model.

The rest of the paper is organised as follows: Section 2 presents a short overview of the most important scenario-generation methods. Section 3 then describes the terminology and notation for the paper. Section 4 provides two criteria for the quality of a scenario tree, and Section 5 shows how to test them. Section 6 then demonstrates the tests on a case from portfolio management. Finally, Section 7 discusses some more aspects of scenario generation, before we conclude the paper.

2 Short overview of scenario-generation methods

2.1 "Pure" Scenario-generation methods

Conditional sampling.

These are the most common methods for generating scenarios. At every node of a scenario tree, we sample several values from the stochastic process $\{\tilde{\boldsymbol{\xi}}_t\}$. This is done either by sampling directly from the distribution of $\{\tilde{\boldsymbol{\xi}}_t\}$, or by evolving the process according to an explicit formula $\tilde{\boldsymbol{\xi}}_{t+1} = z(\boldsymbol{\xi}_t, \tilde{\varepsilon})$, or even $\tilde{\boldsymbol{\xi}}_{t+1} = z(\{\boldsymbol{\xi}_\tau, \tau < t\}, \tilde{\varepsilon})$, sampling from $\tilde{\varepsilon}$.

Traditional sampling methods can sample only from a univariate random variable. When we want to sample a random vector, we need to sample every marginal (the univariate component) separately, and combine them afterwards. Usually, the samples are combined all-against-all, resulting in a vector of independent random variables. The obvious problem is that the size of the tree grows exponentially with the dimension of the random vector: if we sample s scenarios for k marginals, we end-up with s^k scenarios.

Another problem is how to get correlated random vectors – a common approach ([23, 13, 31]) is to find the principal components (which are independent by definition) and sample those, instead of the original random variables. This approach has the additional advantage of reducing the dimension, and therefore reducing the number of scenarios.

There are several ways to improve a sampling algorithm. Instead of a "pure" sampling, we may, for example, use integration quadratures or low discrepancy sequences, if appropriate – see [27]. For symmetric distributions, [22] uses an antithetic sampling. Another way to improve a sampling method is to re-scale the obtained tree, to guarantee the correct mean and variance – see [1].

Sampling from specified marginals and correlations.

As mentioned in the previous section, the traditional sampling methods have problems generating multivariate vectors, especially if they are correlated. However, there are sampling-based methods that solve this problem, using various transformations.

In those methods, the user specifies the marginal distributions and the correlation matrix. In general, there is no restriction on the marginal distributions, they may even be from different families. Examples of such methods can be found in [2, 24, 6].

Moment matching.

The methods from the previous section may be used only if we know the distribution functions of the marginals. If we do not know them, we may describe the marginals by their moments (mean, variance, skewness, kurtosis etc.) instead. In addition, we specify the correlation matrix and possibly—if the method allows us—other statistical properties (percentiles, higher co-moments, etc). Then we *construct* a discrete distribution satisfying those properties. Examples of this approach include [32, 30, 24, 15, 22, 25, 12, 16].

Path-based methods.

These methods start by generating complete paths, i.e. the scenarios, by evolving the stochastic process $\{\tilde{\xi}_t\}$. The result of this step is not a scenario tree, but a set of paths, also called a "fan". To transform a fan to a scenario tree, the scenarios have to be clustered (bound) together, in all-but-the-last period. This process is called clustering or bucketing. Examples of these methods can be found in [8, 17].

"Optimal discretization".

[28] describes a method that tries to find an approximation of a stochastic process (i.e. scenario tree) that minimizes an error in the objective function of the optimization model. Unlike the methods from the previous sections, the whole multi-period scenario tree is constructed at once. On the other hand, it works only for univariate processes. We use some of the methodology from [28] in Section 4.

2.2 Related methods

Scenario reduction.

This is a method for decreasing the size of a given tree. This method tries to find a scenario subset of prescribed cardinality, and a probability measure based on this set, that is closest to the initial distribution in terms of some probability metrics. The method is described in [9, 29].

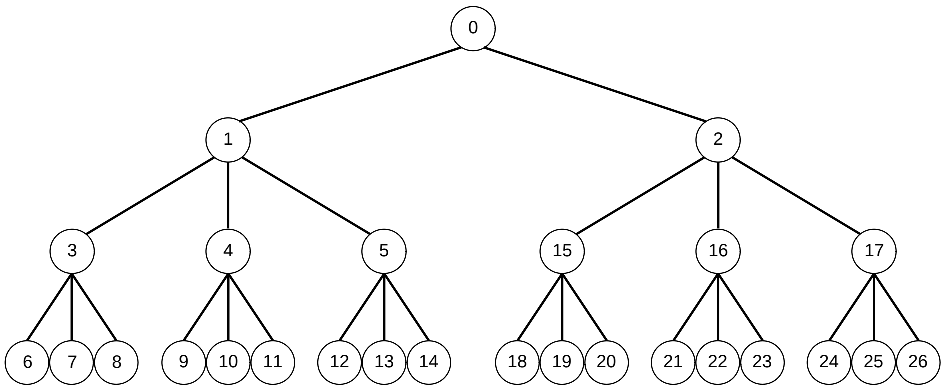

Figure 1: Example of a three-period tree

Internal sampling methods.

Instead of using a pre-generated scenario tree, some methods for solving stochastic programming problems sample the scenarios during the solution procedure. The most important methods of this type are: *stochastic decomposition* [14], importance sampling within Benders' (L-shaped) decomposition [5, 19, 18], and *stochastic quasigradient* methods [10, 11].

In addition, there are methods that proceed iteratively: they solve the problem with the current scenario tree, add or remove some scenarios and solve the problem again. Hence, at least in principle, the scenarios are added exactly where needed. The methods differ in the way they decide where to add/remove the scenarios: [3] uses dual variables from the current solution, while [7] measures the "importance of scenarios" by EVPI (expected value of perfect information).

3 Notation and terminology

Throughout the paper, we use the following conventions: stochastic variables are denoted by tilde (as in $\tilde{\xi}$), and discrete stochastic variables by breve ($\breve{\xi}$). Stochastic processes are described as $\{\tilde{\xi}_t\}_{t\in\mathbf{T}}$, or only $\{\tilde{\xi}_t\}$. The notation can combine, so $\{\breve{\boldsymbol{\xi}}_t\}$ denotes a discrete multivariate process.

Let us have a stochastic programming model with uncertainty described by a stochastic process $\{\tilde{\boldsymbol{\xi}}_t\}_{t\in\mathbf{T}}$. To be able to approximate the process by a scenario tree, the process has to be discrete in time, i.e. $\mathbf{T} = \{0,\ldots,T\}$. We call the points in time $t \in \mathbf{T}$ *stages*.[1] Since choosing the stages is often a natural part of the modelling process, we assume that the time discretization has already been done, so that we have the set $\mathbf{T}$.

In a scenario tree, the "true" stochastic process $\{\tilde{\boldsymbol{\xi}}_t\}$ is approximated by a discrete process $\{\breve{\boldsymbol{\xi}}_t\}$. Since there is a unique relation between the scenario tree and the process $\{\breve{\boldsymbol{\xi}}_t\}$, we often refer to a "$T$-period scenario tree $\{\breve{\boldsymbol{\xi}}_t\}$". For example, the three-period tree in Figure 1 represents a stochastic process with two outcomes in the first period, and three outcomes per node in the last two periods.

In the rest of the paper, we focus on the objective function of the stochastic programming model (1). To simplify the formulas, we denote the whole model by

$$\min_{x\in\mathbf{X}} F\big(\boldsymbol{x};\tilde{\boldsymbol{\xi}}_t\big) \,, \tag{2}$$

[1] There is no general agreement on what should be called stages: in some contexts, stages are only those points in time where a decision is made.

where $\tilde{\boldsymbol{\xi}}_t$ is to be understood as $\{\tilde{\boldsymbol{\xi}}_t\}$. When we approximate the process $\{\tilde{\boldsymbol{\xi}}_t\}$ by a scenario tree $\{\breve{\boldsymbol{\xi}}_t\}$, the objective function becomes $F\big(\boldsymbol{x}; \breve{\boldsymbol{\xi}}_t\big)$.

4 Measure of quality of a scenario tree

We should always remember that our goal is to solve a stochastic program. The only reason why we need a scenario tree is that we do not know how to solve the problem directly with the process $\{\tilde{\boldsymbol{\xi}}_t\}$. Hence, we should judge a scenario tree (and, consequently, a scenario-generation method) by the quality of the decision it gives us. *We are not concerned about how well the distribution is approximated, as long as the scenario tree leads to a "good" decision.* In other words, we are not necessarily searching for a discretization of a distribution that is optimal (or even good) in the statistical sense. See [30] for discussion and examples of this topic.

The error of approximating a stochastic process $\{\tilde{\boldsymbol{\xi}}_t\}$ by a discretization $\{\breve{\boldsymbol{\xi}}_t\}$, for a given stochastic programming problem (2), is thus defined as the difference between the value of the true objective function at the optimal solutions of the true and the approximated problems. The following definition of the error is from [28]:

$$\begin{aligned} e_f(\tilde{\boldsymbol{\xi}}_t, \breve{\boldsymbol{\xi}}_t) &= F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F\big(\boldsymbol{x}; \breve{\boldsymbol{\xi}}_t\big); \tilde{\boldsymbol{\xi}}_t\Big) - F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F\big(\boldsymbol{x}; \tilde{\boldsymbol{\xi}}_t\big); \tilde{\boldsymbol{\xi}}_t\Big) \\ &= F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F\big(\boldsymbol{x}; \breve{\boldsymbol{\xi}}_t\big); \tilde{\boldsymbol{\xi}}_t\Big) - \min_{\boldsymbol{x}} F\big(\boldsymbol{x}; \tilde{\boldsymbol{\xi}}_t\big) \end{aligned} \tag{3}$$

Note that $e_f(\tilde{\boldsymbol{\xi}}_t, \breve{\boldsymbol{\xi}}_t) \geq 0$, since the second element is the true minimum, while the first one is a value of the (true) objective function at an approximate solution. Note also that we do not compare the optimal solutions $\boldsymbol{x}$, but their corresponding values of the objective function. The reason is that the objective function of a stochastic programming problem is typically flat, so there can be different solutions giving very similar objective values.[2]

Definition (3) has one rather obvious problem: the error is, in most practical problems, impossible to calculate. [28] solves this by proving that, under certain uniform Lipschitz conditions,

$$e_f(\tilde{\boldsymbol{\xi}}_t, \breve{\boldsymbol{\xi}}_t) \leq 2 \sup_{\boldsymbol{x}} \Big| F\big(\boldsymbol{x}; \breve{\boldsymbol{\xi}}_t\big) - F\big(\boldsymbol{x}; \tilde{\boldsymbol{\xi}}_t\big) \Big| \leq 2L\, d(\breve{\boldsymbol{\xi}}_t, \tilde{\boldsymbol{\xi}}_t) \ ,$$

where L is a Lipschitz constant of $F()$,[3] and $d(\breve{\boldsymbol{\xi}}_t, \tilde{\boldsymbol{\xi}}_t)$ is a Wasserstein (transportation) distance of the distribution functions of the processes $\{\breve{\boldsymbol{\xi}}_t\}$ and $\{\tilde{\boldsymbol{\xi}}_t\}$. An algorithm is then developed to construct a scenario tree that minimizes the upper bound, i.e. the Wasserstein distance $d(\breve{\boldsymbol{\xi}}_t, \tilde{\boldsymbol{\xi}}_t)$.

This approach has several shortcomings: The bounds can, in general, be quite loose, so even if we find a scenario tree that minimizes the upper bound, there is no guarantee that we will be close to the minimum of $e_f()$. In addition, minimization of the upper bound does not depend on the optimization problem, so we have missed the link between the scenario generation and the problem. (Only the constant L, i.e. the tightness of the bound, depends on the problem.)

In this paper, we have therefore taken a different approach: instead of trying to find the optimal scenario-generation method, we focus on evaluation of a given method. In this context, a scenario-generation method may be seen as a heuristic for minimizing the error $e_f()$, as opposed to [28], which comes with an exact method for minimizing an upper bound of $e_f()$.

There are two problematic operations in definition (3) of the error $e_f(\tilde{\boldsymbol{\xi}}_t, \breve{\boldsymbol{\xi}}_t)$:

[2] In addition, we would need to define a meaningful metric on the space of $\boldsymbol{x}$, which could itself be a problem.

[3] Actually, L is a Lipschitz constant of $f()$, where $F(\boldsymbol{x}, \tilde{\boldsymbol{\xi}}) = \mathbb{E}^{\tilde{\boldsymbol{\xi}}}\big[f(\boldsymbol{x}, \tilde{\boldsymbol{\xi}})\big]$. See [28] for details.

1. finding the “true” objective value $F\big(\boldsymbol{x};\breve{\boldsymbol{\xi}}_t\big)$ for a given solution $\boldsymbol{x}$.

2. finding the “true” optimal solution to (2): $\operatorname{argmin}_{\boldsymbol{x}} F\big(\boldsymbol{x};\tilde{\boldsymbol{\xi}}_t\big)$

While the second is almost always prohibitive, since it needs solving the optimization problem with the continuous process, the first one may be possible, for example via simulation. In the next section, we discuss different approaches for testing the discretization error, together with other tests of the quality of the discretization.

5 Testing a scenario-generation method

There are (at least) two minimal requirements a scenario-generation method must satisfy. Since most of the methods involve some randomness, the first requirement is stability: if we generate several trees (with the same input) and solve the optimization problem with these trees, we should get the same optimal value of the objective function. The other requirement is that the scenario tree should not introduce any bias, compared to the true solution.

There is a conceptual difference between the two requirements: while the first one can, at least to some degree, be tested, a direct testing of the second is in most cases impossible.

5.1 Stability requirement

This requirement can be stated as follows: If we generate several scenario trees (discretizations $\{\breve{\boldsymbol{\xi}}_t\}$) for a given process $\{\tilde{\boldsymbol{\xi}}_t\}$, and solve the stochastic programming problem with each tree, we should get (approximately) the same optimal value of the objective function.

Let us say that we generate K scenario trees $\breve{\boldsymbol{\xi}}_{tk}$, solve the optimization problem with each one of then, and obtain optimal solutions $\boldsymbol{x}^*_k,\ k = 1\dots K$. By an *in-sample stability* we then understand

$$F\big(\boldsymbol{x}^*_k;\breve{\boldsymbol{\xi}}_{tk}\big) \approx F\big(\boldsymbol{x}^*_l;\breve{\boldsymbol{\xi}}_{tl}\big) \qquad k,l \in 1\dots K\ ,$$

while an *out-of-sample stability* is defined as

$$F\big(\boldsymbol{x}^*_k;\tilde{\boldsymbol{\xi}}_t\big) \approx F\big(\boldsymbol{x}^*_l;\tilde{\boldsymbol{\xi}}_t\big) \qquad k,l \in 1\dots K\ .$$

Or, equivalently:

$$\begin{aligned}
&\text{in-sample:} && \min_{\boldsymbol{x}} F\big(\boldsymbol{x};\breve{\boldsymbol{\xi}}_{tk}\big) \approx \min_{\boldsymbol{x}} F\big(\boldsymbol{x};\breve{\boldsymbol{\xi}}_{tl}\big)\\
&\text{out-of-sample:} && F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F\big(\boldsymbol{x};\breve{\boldsymbol{\xi}}_{tk}\big);\tilde{\boldsymbol{\xi}}_t\Big) \approx F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F\big(\boldsymbol{x};\breve{\boldsymbol{\xi}}_{tl}\big);\tilde{\boldsymbol{\xi}}_t\Big)\\
&\text{out-of-sample, using (3):} && e_f(\tilde{\boldsymbol{\xi}}_t,\breve{\boldsymbol{\xi}}_{tk}) \approx e_f(\tilde{\boldsymbol{\xi}}_t,\breve{\boldsymbol{\xi}}_{tl})
\end{aligned}$$

There is an important difference between the two definitions: while for the in-sample stability we need only solve the scenario-based optimization problem, for the out-of-sample stability we have to be able to evaluate the “true” objective function $F\big(\boldsymbol{x};\tilde{\boldsymbol{\xi}}_t\big)$. To be able to do this, we need to have a full knowledge of the distribution of $\{\tilde{\boldsymbol{\xi}}_t\}$, and even then it may not be straightforward to evaluate $F\big(\boldsymbol{x};\tilde{\boldsymbol{\xi}}_t\big)$.

It is important to realize that the two stabilities are different and that there is no simple relationship between them. This can be demonstrated on the following one-period, one-dimensional example:

$$\min_{x\in\mathbb{R}} F\big(x;\tilde{\xi}\big) = \mathbb{E}^{\tilde{\xi}}\Big[\big(x-\tilde{\xi}\big)^2\Big]$$

This problem can be solved explicitly, for any distribution of $\tilde{\xi}$ (we drop the distribution index):

$$\begin{aligned} F\big(x;\tilde{\xi}\big) &= \mathbb{E}\Big[\big(\tilde{\xi}-x\big)^2\Big] \\ &= \mathbb{E}\Big[\Big(\big(\tilde{\xi}-\mathbb{E}[\tilde{\xi}]\big)+\big(\mathbb{E}[\tilde{\xi}]-x\big)\Big)^2\Big] \\ &= \mathbb{E}\Big[\big(\tilde{\xi}-\mathbb{E}[\tilde{\xi}]\big)^2\Big] + \mathbb{E}\Big[2\big(\tilde{\xi}-\mathbb{E}[\tilde{\xi}]\big)\big(\mathbb{E}[\tilde{\xi}]-x\big)\Big] + \mathbb{E}\Big[\big(\mathbb{E}[\tilde{\xi}]-x\big)^2\Big] \\ &= \mathrm{Var}[\tilde{\xi}] + 0 + \big(x-\mathbb{E}[\tilde{\xi}]\big)^2, \end{aligned}$$

so the optimal solution is

$$\begin{aligned} x^* &= \operatorname*{argmin}_{x\in\mathbb{R}} F\big(x;\tilde{\xi}\big) = \mathbb{E}[\tilde{\xi}] \\ F\big(x^*;\tilde{\xi}\big) &= \min_{x\in\mathbb{R}} F\big(x;\tilde{\xi}\big) = \mathrm{Var}[\tilde{\xi}] \end{aligned}$$

Now, assume we generate sample trees $\breve{\xi}_k,\ k = 1\ldots K$, and get the solutions $x^*_k = \mathbb{E}[\breve{\xi}_k]$. Let us first assume that the scenario-generation method is such that all the samples $\breve{\xi}_k$ have the correct means (i.e. $\mathbb{E}[\breve{\xi}_k] = \mathbb{E}[\tilde{\xi}]$), but the variances are different in all the samples. Hence $F\big(x^*_k;\breve{\xi}_k\big) = \mathrm{Var}[\breve{\xi}_k]$ is different for all the samples, so we do not have in-sample stability. At the same time, $x^*_k = x^*$, so $F\big(x^*_k;\tilde{\xi}\big) = F\big(x^*;\tilde{\xi}\big)$, and the out-of-sample stability holds.

If we instead assume that we have a scenario-generation method that produces samples with correct variances (i.e. $\mathrm{Var}[\breve{\xi}_k] = \mathrm{Var}[\tilde{\xi}]$), but the means are different in all the samples,[4] we would have $F\big(x^*_k;\breve{\xi}_k\big) = \mathrm{Var}[\breve{\xi}_k] = \mathrm{Var}[\tilde{\xi}]$, so we would have the in-sample stability. On the other hand, $F\big(x^*_k;\tilde{\xi}\big) = \mathrm{Var}[\tilde{\xi}] + \big(\mathbb{E}[\breve{\xi}_k] - \mathbb{E}[\tilde{\xi}]\big)^2$ would be different for all the samples, so the problem would be out-of-sample unstable.

We may ask what is the practical difference between the in-sample and out-of-sample stability, and which of them is more important to have. Having out-of-sample stability means that the real performance of the solution $\boldsymbol{x}^*_k$ is stable, i.e. it does not depend on which scenario tree $\{\breve{\boldsymbol{\xi}}_t\}$ we choose. However, if we do not have the in-sample stability as well, we may be getting good solutions, but without knowing how good they really are (unless we solve several instances and take an average, or do the out-of-sample evaluation). The opposite (in-sample without out-of-sample stability) is even more dangerous, since the real performance of the solutions depends on which scenario tree we pick—without the possibility of detecting it by solving the problem on several trees.

In the example above, we could see that it is possible to have an in-sample instability in the objective function, but still have an in-sample stability of the solutions—in our case, the solutions were the same in all the sample trees. This obviously guarantees an out-of-sample stability. Therefore, if we detect an in-sample instability of the objective, we should look at the solutions as well. However, it does not work the other way around, i.e. we can have the out-of-sample stability even if the in-sample solutions vary, because the objective functions of stochastic programming problems are typically flat.

It can be expected that in most practical applications we will have either both the stabilities or none, so the in-sample tests should be sufficient in detecting a possible instability. However, if there is a way to perform the out-of-sample test, we would recommend to do that as well.

[4] This may not be a very realistic example, but that is not the point here.

There are several possible ways to do the out-of-sample testing, i.e. the *evaluation* of the objective function $F\big(\boldsymbol{x}_k;\tilde{\boldsymbol{\xi}}_t\big)$ for a given decision $\boldsymbol{x}_k$. If we know the true stochastic process $\{\tilde{\boldsymbol{\xi}}_t\}$, the obvious choice is some Monte-Carlo-like simulation method. If we, on the other hand, use historical data in the scenario generation, back-testing may be an appropriate option. Or, if we have another scenario-generation method we believe to be stable, we may use it to create a reference scenario tree and evaluate the solutions $\boldsymbol{x}_k$ on that tree—notice that the tree can be quite big, since we are not solving a stochastic programming problem on it, we are only evaluating the objective function for a given decision.

To conclude the section, we would like to repeat that stability is the minimal requirement we should put on a scenario-generation method. Hence, before we start to work with a new optimization model, or a new scenario-generation method (remember that we test the two together), we should always run the stability tests: the in-sample test and, if feasible, the out-of-sample test.

5.2 Testing for a possible bias

In addition to being stable (both in-sample and out-of-sample), the scenario-generation method should not introduce any bias into the solution. In other words, the solution of the scenario-based problem,

$$\breve{\boldsymbol{x}}^* = \operatorname*{argmin}_{\boldsymbol{x}} F\big(\boldsymbol{x};\breve{\boldsymbol{\xi}}_t\big) \ ,$$

should be an (almost) optimal solution of the original problem (2). Hence, the value of the "true" objective function at the scenario solution, $F\big(\breve{\boldsymbol{x}}^*;\tilde{\boldsymbol{\xi}}_t\big)$, should be (approximately) equal to the "true" optimal value $\min_{\boldsymbol{x}} F\big(\boldsymbol{x};\tilde{\boldsymbol{\xi}}_t\big)$:

$$F\big(\breve{\boldsymbol{x}}^*;\tilde{\boldsymbol{\xi}}_t\big) = F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F\big(\boldsymbol{x};\breve{\boldsymbol{\xi}}_t\big);\tilde{\boldsymbol{\xi}}_t\Big) \approx \min_{\boldsymbol{x}} F\big(\boldsymbol{x};\tilde{\boldsymbol{\xi}}_t\big) \ .$$

Or, using the definition (3),

$$e_f(\tilde{\boldsymbol{\xi}}_t,\breve{\boldsymbol{\xi}}_t) \approx 0 \ .$$

The problem is that testing of this property is in most practical problems impossible, since it needs solving the optimization problem with the (true) continuous process—and if we could solve that, we would not need scenario trees in the first place.

In some cases, however, it can be possible to do some approximate tests. One possibility is to built a *reference tree*, and use it as a representation (approximation) of the true stochastic process. Typically, such a tree should be as big as possible, i.e. the biggest tree for which we can still solve the optimization problem. To create such a tree, we would need a method that is guaranteed to be unbiased—we can not use the method we want to test! For example, if we use a data series as an input for the scenario generation, we may try using all the history as scenarios.

5.3 Improving the performance

When the testing shows that our scenario-generation method is instable or biased (for the given stochastic programming model), the next question is what are the possible causes of the problem. The answer depends to a large degree on the type of the scenario-generation method used:

Sampling methods.

When we use a sampling method, the strongest candidate for the source of the instability or bias is a lack of scenarios—we know that, with an increasing number of scenarios, the discrete distribution converges to the true distribution. Hence, by increasing the number of scenarios, the trees will be closer to the true distribution, and consequently also closer to each other. As a result, both the instability and the bias should decrease.

In addition to increasing the number of scenarios (which is usually limited by the solution time for the optimization model), we can also try to improve the sampling method. Some of the options are included in the overview in Section 2.

Moment-matching methods.

With moment-matching methods, the situation is more complicated. Since these methods generally do not guarantee convergence, increasing the number of scenarios is not guaranteed to help. We thus need to look at different issues. In the following discussion we assume that in all the tested scenario trees $\breve{\boldsymbol{\xi}}_{tk}$, we have managed to match all the required properties perfectly, i.e. the instability/bias has to come from some properties we do not control (and that can, thus, vary between the tested trees).

Even without the convergence guarantee, the first thing to test is still the number of scenarios: There is an obvious difference between a discrete distribution with three points, and a discrete distribution with thousand points, even if their first four (or even five) moments can be equal. The difference is in the *smoothness* of the distribution, and our experience shows that this is often an important factor. In this context, it is important to understand that not all the moment-matching methods show increasing smoothness with increasing number of scenarios: while this is typically the case for transformation-based methods (for example [16]), it is not true for optimization-based methods like [15].

The important issue of the moment-matching methods is whether we match the right properties—an issue that is obviously dependent on the optimization model. While for a mean-variance model it is enough to match the means, variances, and the correlation matrix, most optimization models will require more. Our experience shows that the first four moments are often a good enough description of the marginals, at least for financial models. On the other hand, a correlation matrix may not be enough to describe the multi-variate structure. In such a case, we may try to match also higher co-moments, or use a copula ([26, 4]), if we have the necessary data and a scenario-generation method that can work with these properties.

What shall we then do, when we discover that a moment-matching method is either instable or biased? The first thing to try is to increase the number of scenarios as much as possible (we still have to be able to solve the optimization model in a reasonable time). If this helps, the problems were probably caused by the lack of smoothness in the original trees. Otherwise, it means that there is some property the decision model reacts to, but we do not control it in the scenario-generation process. We have no general advices on identifying the missing property—it depends on the decision model, and is typically done by a trial-and-error approach, based on problem understanding.

6 Test case: a portfolio optimization

As a test case, we use a simple one-period portfolio optimization problem: we consider one-month investments in three indices (stocks, short-term bonds, and long-term bonds), in four markets (USA, UK, Germany, Japan), giving us twelve assets in total. We model the situation of a US investor, so we have to include the exchange rates of the three foreign currencies to USD. Hence, we have fifteen random variables in the scenario trees. In the model, we do not allow short positions. In addition, it is possible to hedge the currency risk with forward contracts.

As an objective function, we use the expected return and quadratic penalties for shortfalls (returns under a given threshold):

$$\begin{aligned} \mathrm{sf}(\boldsymbol{\xi}) &= \max\left(\mathrm{Tg} - \mathrm{ret}(\boldsymbol{x}, \boldsymbol{\xi}), 0\right) \\ F\big(\boldsymbol{x}; \breve{\boldsymbol{\xi}}\big) &= \mathbb{E}\left[\mathrm{ret}(\boldsymbol{x}, \breve{\boldsymbol{\xi}}) - \alpha\left(\mathrm{sf}(\breve{\boldsymbol{\xi}}) + \beta\, \mathrm{sf}(\breve{\boldsymbol{\xi}})^2\right)\right] \ , \end{aligned}$$

where α is a risk-aversion parameter, and β is a weight of the quadratic term. In the test, we used the following values: $\mathrm{Tg} = 0$, $\alpha = 1$, and $\beta = 10$.

We use the moment-matching scenario-generation method from [16] to generate the scenarios. This method generates scenario trees with specified first four moments of the marginal distributions (mean, standard deviation, skewness and kurtosis), and correlation matrix.

For the out-of-sample test, and for the test of a bias, we need a representation of the “real world”. In our case, we take a large scenario set— that we refer to as the “benchmark scenario set”. *It is important that the benchmark set is provided exogenously, that is, it is not generated by the same method which we want to test.* In our case, the benchmark scenario set (tree) was generated by a method based on principal component analysis described in [31]. The benchmark tree has $20,000$ scenarios, and is based on data in the period from January 1990 to April 2001. See [21] for a detailed description of properties of the benchmark scenario set. We note that the scenarios of the benchmark tree are not equiprobable.

Based on the benchmark tree, we compute the moments and correlations of the differentials of the random variables. The values of these statistical properties constitute the targets to match with our scenario generation procedure.

Since we have the benchmark scenario tree as an representation of the true distribution, we can perform all the tests from Section 5: For a given size of the tree, we generate 25 scenario trees, solve the optimization model on each of them, and then evaluate the solutions on the benchmark tree. This is repeated for several different sizes of the tree.

Results of the test are presented in Table 1. We see that the scenario-generation method used gives a reasonable stability, both in-sample and out-of-sample. We see also that the out-of-sample values have a smaller variance then the in-sample values. The reason is that in in-sample tests we evaluate (different) solutions on different trees, while in the out-of-sample tests we evaluate all the solutions on the common benchmark tree. Note also that the performance (true objective value of the solutions) improves as the number of scenarios increases.

Another important observation is the fact that, in the case of 50 scenarios, the in-sample objective values are significantly higher than the out-of-sample (true) values. In other words, the solution is notably worse than the model tells us. This is a common observation: when we do not have enough scenarios, the model overestimates the quality of its own solution. Only out-of-sample evaluations can tell us how good a solution really is.

In addition to the stability tests, we have solved the optimization model on the benchmark tree, and obtained the “true” optimal solution: 0.00930. Hence, we see that the scenario generation method does not introduce any significant bias, given there are enough scenarios. We also see that there is a noticeable bias in the case of 50 scenarios.

The conclusion of the tests is that the tested scenario-generation method is suitable for the given optimization model: it is stable and does not introduce any significant bias, provided we have enough scenarios. The results also suggest that we should not use trees smaller than 1000 scenarios.

Table 1: Stability tests for the optimization model. For every size of the scenario tree, 25 trees were generated, and the model was solved on each of them. The solutions were then evaluated on the benchmark tree to obtain the out-of-sample values. The table presents sample means and standard deviations of the optimal values, for the different sizes.

description of the test			# of scenarios			
type of test	objective f.	value	50	250	1000	5000
in-sample	$F(\boldsymbol{x}_k^*; \breve{\boldsymbol{\xi}}_{tk})$	average	0.00948	0.00936	0.00931	0.00929
		std. dev.	0.00023	0.00011	0.00005	0.00002
out-of-sample	$F(\boldsymbol{x}_k^*; \tilde{\boldsymbol{\xi}}_t)$	average	0.00902	0.00926	0.00928	0.00930
		std. dev.	0.00015	0.00003	0.00001	0.00000

7 How far can we get?

So far, we have implicitly assumed that all distributions are known. In reality this is very rarely the case. What does this lack of knowledge mean, particularly for the issue we raise here, that of generating good scenario trees? First, let us distinguish between three cases:

- The distributions are fully known.
- We have theoretical knowledge about the distribution family, plus data.
- We only have data.

Although it is common to assume in theoretical papers that a distribution is fully known—very often this is done indirectly by basing the paper (or the tests within the paper) on certain distributions—this is in our view not the case in many applications. An exception may be planning under well-known stochasticity, such as the roll of a (fair) die or the flip of a (fair) coin. But most interesting applications concern real-life phenomena such as price, demand or quality. So, if the distribution is not known, what can we then say about scenario trees? The most important, and also obvious, observation is that certain theoretical properties of scenario generation methods become less useful. For example, sample-based methods will, if the sample is large enough, produce scenario trees arbitrarily close to the distribution from which we sample. But how useful is it to know that we have convergence towards something that is not the real thing?

This becomes even more important if we have do not know a distribution family. A common approach in this case is to assume some family. If we do so, we *know* that sampling will converge to a distribution that contains information we have added without foundation in data or theory. An alternative is using the empirical distribution directly. In this way we avoid adding any subjective information to the data. On the other hand, this approach will prevent the use of any methodology that requires knowledge of the distribution itself.

In the (scarce) occasions that we have distributional information, it is normally advisable to estimate the distribution, since otherwise that information is lost. But we still face the problem of having convergence to a distribution that may not be the right one.

But there is more to the problem than this. To use data we have to assume that the past is a reasonable description of the future. Whether this is the case cannot be tested, it is a question of belief. We can of course test, at least in many cases, whether or not the past would have been a good description of the future, in the past. But it is still a question of belief if this is the case now.

We have mentioned simulation as a way to evaluate the true value of a certain decision. The good thing about simulation models is that they can be allowed to contain details that we are unable to put into the optimization model. But all the problems discussed above remain. Within the simulation model we need to sample from distributions, and they are normally not fully known.

This discussion may seem to be very negative, we seem to be saying that nothing works. That is not the point. The point is to be aware of the shortcomings of modeling in general, and stochastic programming in particular. We can get to a certain point, but thereafter empirical testing becomes impossible, and we have to start believing in what we do. And in our view, this also means that we should be very sceptical to high accuracy statements from models. We may know that a given method retains two digits of accuracy, but we cannot know how many correct digits were in the input.

Hence, the convergence properties are not so important in real-life applications. Instead, it is more important to have scenario generation tools that give us reasonable control over the tree with a limited number of scenarios. What we want is a method that is stable, unbiased and produces small trees. But there is a limit to what we require in terms of accuracy, given these properties.

Conclusions

In this paper, we have discussed how to evaluate the suitability of a given scenario-generation method for a given stochastic programming problem. We have identified the main properties the scenario-generation method should satisfied, and suggested a way to test them. We have also demonstrated the methodology on an example from portfolio management.

References

[1] D. R. Cariño, T. Kent, D. H. Myers, C. Stacy, M. Sylvanus, A. L. Turner, K. Watanabe, and W. T. Ziemba. The Russell-Yasuda Kasai model: an asset liability model for a Japanese insurance company using multistage stochastic programming. *INTERFACES*, 24(1):29–49, 1994.

[2] M. C. Cario and B.L. Nelson. Modeling and generating random vectors with arbitrary marginal distributions and correlation matrix. Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois, 1997.

[3] Michael Casey and Suvrajeet Sen. The scenario generation algorithm for multistage stochastic linear programming. Available at `http://www.math.ups.edu/~mcasey/`, 2002.

[4] Robert T. Clemen and Terence Reilly. Correlations and copulas for decision and risk analysis. *Management Science*, 45(2):208–224, February 1999.

[5] George B. Dantzig and Gerd Infanger. Large-scale stochastic linear programs—importance sampling and Benders decomposition. In *Computational and applied mathematics, I (Dublin, 1991)*, pages 111–120. North-Holland, Amsterdam, 1992.

[6] B. Deler and B. L. Nelson. Modeling and generating multivariate time series with arbitrary marginals using an autoregressive technique. Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois, 2000.

[7] M. A. H. Dempster and R. T. Thompson. EVPI-based importance sampling solution procedures for multistage stochastic linear programmes on parallel MIMD architectures. *Annals of Operations Research*, 90:161–184, 1999. also in Proceedings of the POC96 Conference, Versailles, March, 1996.

[8] Jitka Dupačová, Giorgio Consigli, and Stein W. Wallace. Scenarios for multistage stochastic programs. *Ann. Oper Res.*, 100:25–53 (2001), 2000.

[9] Jitka Dupačová, Nicole Gröwe-Kuska, and Werner Römisch. Scenario reduction in stochastic programming. *Mathematical Programming*, 95(3):493–511, 2003.

[10] Yu. Ermoliev. *Methods of Stochastic Programming*. Nauka, Moscow, 1976. In Russian.

[11] Yury M. Ermoliev and Alexei A. Gaivoronski. Stochastic quasigradient methods for optimization of discrete event systems. *Ann. Oper. Res.*, 39(1-4):1–39 (1993), 1992.

[12] N. Gülpınar, B. Rustem, and R. Settergren. Optimisation and simulation approaches to scenario tree generation. *Journal of Economics Dynamics and Control*, to appear, 2002.

[13] Roger Halldin. *Scenario Trees for Inflow Modelling in Stochastic Optimisation for Energy Planning*. PhD thesis, Lund Unversity, Sweden, 2002.

[14] J. L. Higle and S. Sen. Stochastic decomposition: An algorithm for two-stage linear programs with recourse. *Mathematics of Operations Research*, 16:650–669, 1991.

[15] K. Høyland and S. W. Wallace. Generating scenario trees for multistage decision problems. *Management Science*, 47(2):295–307, 2001.

[16] Kjetil Høyland, Michal Kaut, and Stein W. Wallace. A heuristic for moment-matching scenario generation. *Computational Optimization and Applications*, 24(2-3):169–185, 2003.

[17] IBM Corp. *IBM Optimization Library Stochastic Extensions Users Guide*, 1998.

[18] G. Infanger. *Planning under Uncertainty: Solving Large-Scale Stochastic Linear Programs*. Boyd and Fraser, Danvers, 1994.

[19] Gerd Infanger. Monte Carlo (importance) sampling within a Benders decomposition algorithm for stochastic linear programs. *Ann. Oper. Res.*, 39(1-4):69–95 (1993), 1992.

[20] P. Kall and S.W. Wallace. *Stochastic Programming*. Wiley, Chichester, 1994.

[21] Michal Kaut, Stein W. Wallace, Hercules Vladimirou, and Stavros Zenios. Stability analysis of a portfolio management model based on the conditional value-at-risk measure. Feb 2003.

[22] R. R. P. Kouwenberg. Scenario generation and stochastic programming models for asset liability management. *European Journal of Operational Research*, 134(2):51–64, 2001.

[23] Mico Loretan. Generating market risk scenarios using principal components analysis: Methodological and practical considerations. In *The Measurement of Aggregate Market Risk, CGFS Publications No. 7*, pages 23–60. Bank for International Settlements, November 1997. Available at `http://www.bis.org/publ/ecsc07.htm`.

[24] P. M. Lurie and M. S. Goldberg. An approximate method for sampling correlated random variables from partially-specified distributions. *Management Science*, 44(2):203–218, 1998.

[25] Johan Lyhagen. A method to generate multivariate data with moments arbitrary close to the desired moments. Working paper 481, Stockholm School of Economics, December 2001.

[26] Roger B. Nelsen. *An Introduction to Copulas*. Springer-Verlag, New York, 1998.

[27] T. Pennanen and M. Koivu. Integration quadratures in discretization of stochastic programs. Stochastic Programming E-Print Series, `http://www.speps.info`, May 2002.

[28] G. C. Pflug. Scenario tree generation for multiperiod financial optimization by optimal discretization. *Mathematical Programming*, 89(2):251–271, 2001.

[29] W. Römisch and H. Heitsch. Scenario reduction algorithms in stochastic programming. *Computational Optimization and Applications*, 24(2-3):187–206, 2003.

[30] James E. Smith. Moment methods for decision analysis. *Management Science*, 39(3):340–358, March 1993.

[31] N. Topaloglou, Vladimirou H., and S. A. Zenios. CVaR models with selective hedging for international asset allocation. *Journal of Banking and Finance*, 26(7):1535–1561, 2002.

[32] C. David Vale and Vincent A. Maurelli. Simulating multivariate nonnormal distributions. *Psychometrika*, 48(3):465–471, 1983.