Optimization under uncertainty: modeling and solution methods

Paolo Brandimarte
Dipartimento di Scienze Matematiche
Politecnico di Torino

e-mail: paolo.brandimarte@polito.it
URL: http://staff.polito.it/paolo.brandimarte

Lecture 5: Scenario generation for stochastic programming
REFERENCES

1. **Motivation**

2. **Crude Monte Carlo methods**

3. **Improving scenario generation**

4. **Stability in stochastic programming**
THE NEED FOR SCENARIO GENERATION

The problem involving the “true” distribution of random variables,

\[
\begin{align*}
\min & \quad c^T x + E[q(\xi)^T y(\xi)] \\
\text{s.t.} & \quad Ax = b \\
& \quad Wy(\xi) + T(\xi)x = h(\xi), \quad \forall \xi \in \Xi \\
& \quad x, y(\xi) \geq 0
\end{align*}
\]

is typically intractable, either because the distribution is continuous, or because there are too many scenarios.

Then, we consider the discretized, or reduced, problem:

\[
\begin{align*}
\min & \quad c^T x + \sum_{s \in S} \pi_s q_s^T y_s \\
\text{s.t.} & \quad Ax = b \\
& \quad Wy_s + T_s x = h_s, \quad \forall s \in S \\
& \quad x, y_s \geq 0
\end{align*}
\]
A few algorithms (stochastic decomposition and stochastic dual dynamic programming) interleave scenario generation (sampling) and optimization. They generate statistical estimates of cutting planes and test optimality conditions statistically.

Here we just consider simpler methods that generate one scenario tree before optimization, within a two-stage setting.

Even so, we have to deal with difficult issues:

- How can we generate scenarios?
- How many scenarios should we include?
- How can we check the quality of the solution we obtain?

We should *not* take for granted that Monte Carlo sampling, the seemingly obvious choice, is the best method for scenario generation.
Numerical integration

Monte Carlo methods should be regarded as numerical integration methods. They can be used to estimate the expected value of a function $g(\cdot)$ of a random variable $X$ with probability density $f_X(x)$:

$$E[g(X)] = \int_{-\infty}^{+\infty} g(x) f_X(x) \, dx.$$  

Note that probabilities, too, can be regarded as expected values:

$$P(A) = \int_{-\infty}^{+\infty} I_A(x) f_X(x) \, dx,$$

where $I_A(x)$ is the indicator function for event $A$.

In stochastic optimization, we are interested in a function defined by an expectation:

$$H(z) = E_X[g(X, z)] = \int_{-\infty}^{+\infty} g(x, z) f_X(x) \, dx.$$
Consider the problem of approximating the value of a definite integral like

\[ I[f] = \int_{a}^{b} f(x) \, dx \]

over a bounded interval \([a, b]\) for a function \(f\) of a single variable.

Since the integration is a linear operator, it is natural to look for an approximation preserving this property. Using a finite number of values of \(f\) over a set of nodes \(x_j\) such that

\[ a = x_0 < x_1 < \cdots < x_N = b, \]

we define a quadrature formula, characterized by weights \(w_j\) and nodes \(x_j\):

\[ Q[f] = \sum_{j=0}^{n} w_j f(x_j). \]

The idea does not scale well to high dimensions.
Consider an integral on the unit interval $[0, 1]$: 

$$I = \int_0^1 g(x) \, dx.$$ 

We may think of this integral as the expected value $E[g(U)]$, where $U$ is a uniform random variable on the interval $(0, 1)$, i.e., $U \sim (0, 1)$.

What we have to do is generating a sequence $\{U_i\}$ of independent random observations from the uniform distribution and then evaluate the sample mean:

$$\hat{I}_m = \frac{1}{m} \sum_{i=1}^{m} g(U_i).$$

The strong law of large numbers implies that, with probability 1,

$$\lim_{m \to \infty} \hat{I}_m = I.$$
MC MECHANICS: (PSEUDO-)RANDOM VARIATE GENERATION

The raw material of any Monte Carlo simulation is a stream of independent realizations of a uniform random variable $U(0, 1)$. An array of methods is offered by software tools like R or Matlab.

Then, a transformation is applied in order to sample from another distribution (see Kroese et. al, 2011). The simplest one is the inverse transform method:

1. draw a random number $U \sim U(0, 1)$,
2. return $X = F^{-1}(U)$, where $F(x) = P\{X \leq x\}$ is the CDF of $X$.

Thus, we see that a Monte Carlo simulation is essentially a way to estimate a multidimensional integral of some function on a unit hypercube:

$$\int_{[0,1]^d} \mathcal{F}(\mathbf{x}) \, d\mathbf{x},$$

where $[0, 1]^d = [0, 1] \times [0, 1] \times \cdots \times [0, 1]$. 

P. Brandimarte – Dip. di Scienze Matematiche
We are quite familiar with confidence intervals of the form

$$\bar{X} \pm t_{n-1,1-\alpha/2} \frac{S}{\sqrt{n}}.$$ 

Indeed, convergence of Monte Carlo estimates (assuming that suitable conditions hold) is related to $\sigma/\sqrt{n}$.

Good news: this rate of convergence does not depend on problem dimensionality.

Bad news: this rate of convergence is slow. How many additional replications do we need to gain one order of magnitude in terms of precision?

Brute force certainly does not work with stochastic programming.
Alternative ideas:

- Monte Carlo with variance reduction
- Gaussian quadrature
- Low discrepancy sequences (quasi-Monte Carlo methods)
- Optimized scenario generation by property matching
- Optimal scenario reduction
In order to improve convergence, which depends on $\sigma/\sqrt{n}$, we may try reducing $\sigma$, rather than increasing the sample size. Several variance reduction strategies are available:

- Antithetic sampling
- Common random numbers
- Control variates
- Importance sampling
- Conditional Monte Carlo
- Latin Hypercube sampling
- Stratified sampling

See (Brandimarte, 2006) or (Brandimarte, 2013) for details, and (Higle, 1998) or (Infanger, 1998) for applications to stochastic programming.
In plain Monte Carlo, we generate a sequence of independent observations. However, inducing some correlation in a clever way may be helpful.

Consider the idea of generating a sequence of paired replications \((X_1^{(i)}, X_2^{(i)}), i = 1, \ldots, n:\)

\[
\begin{array}{cccc}
X_1^{(1)} & X_1^{(2)} & \cdots & X_1^{(n)} \\
X_2^{(1)} & X_2^{(2)} & \cdots & X_2^{(n)}
\end{array}
\]

These observations are “horizontally” independent, in the sense that \(X_j^{(i_1)}\) and \(X_k^{(i_2)}\) are independent however we choose \(j, k = 1, 2\), provided \(i_1 \neq i_2\).

Thus, the pair-averaged observations \(X^{(i)} = (X_1^{(i)} + X_2^{(i)})/2\) are independent, and we may build a confidence interval based on them.
The variance of the sample mean $\bar{X}(n)$ based on the observations $X^{(i)}$ is

$$\text{Var}[\bar{X}(n)] = \frac{\text{Var}(X^{(i)})}{n} = \frac{\text{Var}(X_1^{(i)}) + \text{Var}(X_2^{(i)}) + 2 \text{Cov}(X_1^{(i)}, X_2^{(i)})}{4n} = \frac{\text{Var}(X)}{2n} (1 + \rho(X_1, X_2)).$$

In order to reduce variance, we should take negatively correlated replications within each pair.

To induce a negative correlation, we may use a random number sequence $\{U_k\}$ for the first replication in each pair, and then $\{1 - U_k\}$ in the second one.

However, the negative correlation in the input streams need not induce negative correlation in the output, unless some monotonicity condition applies to the whole simulation chain.
Gaussian quadrature

A typical way to measure the quality of a quadrature formula is the order of the polynomials for which the quadrature formula is exact.

In Newton–Cotes formulas we fix nodes and try to find suitable weights so that the order of the formula is as large as possible.

In Gaussian quadrature nodes and weights are jointly determined, in order to improve order.

Gaussian quadrature formulas are associated with non-negative weight function $w(x)$, and we look for a quadrature formula like

$$\int_{a}^{b} w(x)f(x) \, dx \approx \sum_{i=1}^{n} w_i f(x_i), \quad (9)$$

In our setting, $w(x)$ can be interpreted as a probability density. Gauss–Hermite quadrature, where $w(x) = e^{-x^2}$, is clearly connected with computing the expected value of a function of a normal random variable.
GAUSSIAN QUADRATURE

Let $Y$ be a random variable with normal distribution $\mathcal{N}(\mu, \sigma^2)$. Then

$$E[f(Y)] = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}(\frac{y-\mu}{\sigma})^2} f(y) \, dy.$$ 

In order to use weights and nodes from a Gauss–Hermite formula, we need the following change of variable

$$-x^2 = -\frac{1}{2} \left( \frac{y - \mu}{\sigma} \right)^2 \Rightarrow y = \sqrt{2}\sigma x + \mu \Rightarrow \frac{dy}{\sqrt{2}\sigma} = dx.$$ 

Hence

$$E[f(Y)] \approx \frac{1}{\sqrt{\pi}} \sum_{i=1}^{n} w_i f(\sqrt{2}\sigma x_i + \mu).$$

We may interpret the formula as a discretized distribution, with values $x_i$ and probabilities $w_i$. 

P. Brandimarte – Dip. di Scienze Matematiche
Optimization Under Uncertainty
From the properties of the lognormal distribution, we know that if $X \sim \mathcal{N}(\mu, \sigma^2)$, then

$$E\left[ e^X \right] = e^{\mu + \sigma^2/2}$$

Let $\mu = 4$ and $\sigma = 2$, then $e^{\mu + \sigma^2/2} = 403.4288$.

Using a sample size $n = 1000$, crude Monte Carlo in R yields:

```r
> set.seed(55555)
> mean(exp(rnorm(1000,mu,sigma)))
355.9714
> mean(exp(rnorm(1000,mu,sigma)))
387.6861
> mean(exp(rnorm(1000,mu,sigma)))
449.0870
> mean(exp(rnorm(1000,mu,sigma)))
329.0448
```

We observe a lot of variability.
**Gaussian Quadrature: Numerical Example**

The following R script uses the `statmod` package and illustrates the precision achieved with a few nodes:

```r
# script to check Gauss-Hermite quadrature
require(statmod)
mu <- 4
sigma <- 2
numNodes <- c(5,10,15,20)
trueValue <- exp(mu + sigma^2/2)
for (i in 1:length(numNodes)) {
  out <- gauss.quad(numNodes[i], kind="hermite")
  w <- out$weights/sqrt(pi)
  x <- sqrt(2)*sigma*out$nodes + mu
  approxValue <- sum(w*exp(x))
  percError <- 100*abs(trueValue-approxValue)/trueValue
  cat("N=",numNodes[i]," True=",trueValue, " Approx=",
       approxValue, " %error", percError,"
}
```

N=  5  True= 403.4288  Approx= 398.6568  %error 1.182865
N= 10  True= 403.4288  Approx= 403.4286  %error 5.537709e-05
N= 15  True= 403.4288  Approx= 403.4288  %error 1.893283e-10
N= 20  True= 403.4288  Approx= 403.4288  %error 9.863052e-14
Low-discrepancy sequences

A Monte Carlo simulation is the calculation of an integral over a unit hypercube. Hence, we should fill the unit hypercube in the most “uniform” way.

Consider a sequence \( \mathcal{P} = (x^1, x^2, \ldots, x^N) \) of \( N \) points in the \( m \)-dimensional hypercube \( I^m = [0, 1]^m \subset \mathbb{R}^m \).

If the points are “well distributed,” their number included in any subset \( G \) of \( I^m \) should be roughly proportional to its volume \( \text{vol}(G) \).

Given a point \( z = (z_1, z_2, \ldots, z_m) \), consider the rectangular subset \( G_z \)

\[
G_z = [0, z_1) \times [0, z_2) \times \cdots \times [0, z_m),
\]

with volume \( z_1 z_2 \cdots z_m \).

Let \( S_{\mathcal{P}}(G) \) be a function counting the number of points in the sequence \( \mathcal{P} \) that are contained in a subset \( G \subset I^m \).
The **star discrepancy** of the sequence $\mathcal{P}$ is defined as:

$$D(\mathcal{P}) = \sup_{z \in \mathbb{I}^m} \left| \frac{S_{\mathcal{P}}(G_z)}{N} - z_1 z_2 \cdots z_m \right|.$$  

When computing a multidimensional integral on the unit hypercube, it is natural to look for low-discrepancy sequences.

There is a theoretical result suggesting that low-discrepancy sequences may perform better than pseudorandom sequences.

We know that the estimation error with Monte Carlo simulation is something like $O(1/\sqrt{N})$, where $N$ is the sample size.

Sequences achieving a star discrepancy of order $O((\ln N)^m/N)$ are called low-discrepancy.
HALTON SEQUENCES IN MATLAB

The simplest low-discrepancy sequence is the Halton sequence, in which a prime number is associated with each dimension and the corresponding Van der Corput sequence is generated.

This is a sequence in the unit interval $[0, 1]$, and it based on a simple recipe:

- Represent an integer number $n$ in a base $b$, where $b$ is a prime number:

$$n = (\cdots d_4 d_3 d_2 d_1 d_0)_b$$

- Reflect the digits to obtain a number within the unit interval:

$$h = (0.d_0 d_1 d_2 d_3 d_4 \cdots)_b$$

More formally:

$$n = \sum_{k=0}^{\infty} d_k b^k \quad \Rightarrow \quad h(n, b) = \sum_{k=0}^{\infty} d_k b^{-(k+1)}.$$
HALTON SEQUENCES IN MATLAB

This is the Halton sequence with bases (2, 3, 5), using the Statistics Toolbox of Matlab:

```matlab
» pp = haltonset(3);
» net(pp,10)
ans =
    0   0   0
   0.5000  0.3333  0.2000
   0.2500  0.6667  0.4000
   0.7500  0.1111  0.6000
   0.1250  0.4444  0.8000
   0.6250  0.7778  0.0400
   0.3750  0.2222  0.2400
   0.8750  0.5556  0.4400
  0.0625  0.8889  0.6400
  0.5625  0.0370  0.8400
```
Halton vs. Pseudorandom Sequences

The following picture suggests that a Halton sequence provides a more uniform fill than pseudorandom numbers:
Unfortunately, Halton sequences have bad properties when large prime numbers are involved. This is the filling with bases 109 and 113:
To solve the difficulty, Sobol sequences yield a suitable permutation of the Van der Corput with base 2:

```matlab
» pp = sobolset(5);
» net(pp,10)
ans =
    0   0   0   0   0   0
    0.5000 0.5000 0.5000 0.5000 0.5000
    0.2500 0.7500 0.2500 0.7500 0.2500
    0.7500 0.2500 0.7500 0.2500 0.7500
    0.1250 0.6250 0.8750 0.8750 0.6250
    0.6250 0.1250 0.3750 0.3750 0.1250
    0.3750 0.3750 0.6250 0.1250 0.8750
    0.8750 0.8750 0.1250 0.6250 0.3750
    0.0625 0.9375 0.6875 0.4375 0.8125
    0.5625 0.4375 0.1875 0.9375 0.3125
```
**PROPERTY MATCHING**

This is a deterministic scenario generation approach, based on the idea that the discretized distribution should match some properties (e.g., moments) of the original distribution.

Consider a multivariate normal variable $\mathbf{X}$, with expected value $\mu$ and covariance matrix $\Sigma$. We also know that, for each marginal distribution, skewness $\xi_i = \mathbb{E}[(X_i - \mu_i)^3/\sigma_i^3]$ is zero and kurtosis $\chi_i = \mathbb{E}[(X_i - \mu_i)^4/\sigma_i^4]$ is 3.

Say that we want to generate a scenario fan (the single stage of a tree) of size $S$ and that each realization has probability $1/S$, for the sake of simplicity. Let us denote by $x_i^s$ the value of the random variable $i$ in scenario $s$.

Then, we should have:

\[
\frac{1}{S} \sum_s x_i^s \approx \mu_i \quad \forall i
\]
\[
\frac{1}{S} \sum_s (x_i^s - \mu_i)(x_j^s - \mu_j) \approx \sigma_{ij} \quad \forall i, j
\]
\[
\frac{1}{S} \sum_s (x_i^s - \mu_i)^3/\sigma_i^3 \approx 0 \quad \forall i
\]
\[
\frac{1}{S} \sum_s (x_i^s - \mu_i)^4/\sigma_i^4 \approx 3 \quad \forall i.
\]
Approximate moment matching is obtained by solving the problem:

\[
\min \ w_1 \sum_i \left[ \frac{1}{S} \sum_s x_i^s - \mu_i \right]^2 + w_2 \sum_{i,j} \left[ \frac{1}{S} \sum_s \left(x_i^s - \mu_i\right) \left(x_j^s - \mu_j\right) - \sigma_{ij} \right]^2 \\
+ w_3 \sum_i \left[ \frac{1}{S} \sum_s \left(\frac{x_i^s - \mu_i}{\sigma_i}\right)^3 \right]^2 + w_4 \sum_i \left[ \frac{1}{S} \sum_s \left(\frac{x_i^s - \mu_i}{\sigma_i}\right)^4 - 3 \right]^2
\]

The objective function includes four weights \( w_k \) which may be used to fine tune performance.

Also note that this is (in general) a nonconvex problem.
See (Hoyland, Wallace, 2001) and (King, Wallace, 2012) for examples and heuristic strategies for the generation of multistage trees.
Optimal scenario reduction

The idea of optimal scenario reduction is to sample a large set of scenarios, resulting in a probability measure $\mathbb{P}$, and then to reduce the tree, resulting in a probability measure $\mathbb{Q}$.

The optimal reduction should minimize a suitable metric for the distance between probability measures.

For instance let us consider:

- a set of $S$ scenarios $\xi^i = \{\xi^i_t\}_{t=1}^T$ with probabilities $p_i$;
- a set of $\tilde{S}$ scenarios $\tilde{\xi}^j = \{\tilde{\xi}^j_t\}_{t=1}^T$ with probabilities $q_j$. 
The Kantorovich distance $D_K$, in this discrete case, is defined as

$$D_K(P, Q) = \inf \sum_{i=1}^{S} \sum_{j=1}^{\tilde{S}} \eta_{ij} c(\xi^i, \tilde{\xi}^j)$$

s.t.

$$\sum_{i=1}^{S} \eta_{ij} = q_j, \quad \forall j$$

$$\sum_{j=1}^{\tilde{S}} \eta_{ij} = p_i, \quad \forall i$$

$$\eta_{ij} \geq 0,$$

where

$$c(\xi^i, \tilde{\xi}^j) = \sum_{\tau=1}^{T} |\xi^i_{\tau} - \tilde{\xi}^j_{\tau}|$$
**Optimal scenario reduction**

We see that evaluating the distance boils down to solving a transportation problem.

Given this distance, it is possible to build heuristics in which scenarios are successively eliminated, redistributing probability mass in an optimal way.

It is possible to prescribe scenario cardinality or the quality of the approximation.

This idea was implemented in the Scenred package, for use with GAMS. See, e.g., (Heitsch, Roemisch, 2003) for more details.
Motivation

Crude Monte Carlo methods

Improving scenario generation

Stability in stochastic programming

Stability: in-sample and out-of-sample

**Stability: in-sample vs. out-of-sample**

Whatever scenario reduction strategy is employed (deterministic or stochastic) it is important to check the stability of the resulting solution (Dupačová, 1990). There are quite sophisticated studies on stability of stochastic models (e.g., Rachev, 1991). Here we just consider a down-to-Earth approach (King, Wallace, 2012).

Let us use the streamlined notation

$$\min_x f(x; \xi), \quad \min_x f(x; T)$$

\[ \text{to refer to the problem for the “true” probability distribution and the scenario tree, respectively.} \]

In-sample stability means that if we sample trees $T_i$ and $T_j$, leading to solutions $x_i^*$ and $x_j^*$, respectively, we obtain

$$f(x_i^*; T_i) \approx f(x_j^*; T_j)$$

Note that we are not requiring stability in the solution itself, but in the objective value. This is sensible if what matters is cost, profit, etc.
If the scenario tree is generated deterministically, we might compare trees with slightly different branching structures.

Out of sample stability means

\[ f(x^*_i; \xi) \approx f(x^*_j; \xi) \]

In two stage problems, this check is not hard, as it requires solving many second stage problems. In multistage models, a (costly) rolling horizon simulation procedure is needed.

Alternatively, we might check whether

\[ f(x^*_i; T_j) \approx f(x^*_j; T_i) \]