Algorithms for two-stage stochastic linear programmming

Basic Course on Stochastic Programming, IMPA 2016

Description

Consider the following two-stage stochastic linear program

$$\begin{cases} \min\limits_{x} & c^\top x + \sum_{i=1}^{N} p_i [Q(x, \xi^i)] \\ \text{s.t.} & Ax = b \\ & x \geq 0\,, \end{cases} \tag{1a}$$

with $p_i > 0$ the probability of scenario $\xi^i := (q^i, T^i, W^i, h^i)$, and $Q(x, \xi)$ the optimal value of the following linear programming problem (LP):

$$Q(x, \xi) := \begin{cases} \min\limits_{y} & q^\top y \\ \text{s.t.} & Tx + Wy = h \\ & y \geq 0\,. \end{cases} \tag{1b}$$

In the above notation, vectors have the following dimensions: $c \in \mathbb{R}^{n_1}$, $b \in \mathbb{R}^{m_1}$, $q \in \mathbb{R}^{n_2}$ and $h \in \mathbb{R}^{m_2}$. The involved matrices A, T and W have appropriate sizes.

The work consist in coding three algorithms for solving two-stage stochastic linear programs of the form (1). For testing your codes we made available on the course's web page `http://svan2016.sciencesconf.org/resource/page/id/26` a *toy problem* whose dimensions are: $n_1 = n_2 = 60$, $m_1 = 30$ and $m_2 = 40$. Access the link and download the following MATLAB files:

- `ScenData.mat`, containing $100,000$ scenarios for problem (1). To load the data, use the MATLAB command: `load('ScenData.mat')`. A matrix `scen` of scenarios will appear in the MATLAB workspace. Each row of `scen` represents a scenario. Unless explicitly stated, consider the first N rows of `scen` to obtain N scenarios.

- `OutSample.mat`, containing other $100,000$ scenarios for problem (1). This set of scenarios is to be used for the out-of-sample simulation. The data structure is the same as in the previous item.

- `DataPbm.m`. This script returns all the vectors and matrices in problem (1). Type `help DataPbm` in the `Matlab` command window for a description of the script's input and output.

Remarks and advices

The MATLAB solver for linear programming `linprog` will be extensively used in this work. We advice to employ `linprog` with the default parameters. As alternatives to MATLAB and `linprog`, you may use `Octave` and its solver `glpk` for linear programming problem. We also adivice to use default parameters.

Each student should send to `svanprog@impa.br`:

- a report (in pdf format) on this work, containing answers to the questions listed below and all the MATLAB sources in a readable manner[1].
- all the MATLAB sources (in .m format) and a main script called `BAS` gathering all your solvers. See the Appendix for more details.

This is an individual work, however students are allowed to interact with each other to exchange experiences and advices. Codes or reports co-authored by m authors will have the score divided by m. In case of plagiarism the score will be zero.

Exercise 1: equivalent deterministic program

Let N be the number of considered scenarios. Write a MATLAB script to solve the *Equivalent Deterministic* problem:

$$\begin{cases} \min & c^\top x + \sum_{i=1}^N p_i \, {q^i}^\top y^i \\ \text{s.t.} & Ax = b \\ & T^i x + W^i y^i = h^i, \qquad \forall\, i = 1, \dots, N \\ & x \geq 0, y_i \geq 0, \qquad \forall\, i = 1, \dots, N\,. \end{cases} \tag{2}$$

Run your script with $N = 1$ (obtaining the data's problem by the command `[h,q,W,T,A,c,b]= DataPbm (1,0,scen)`) and confirm that the optimal value of (2) is $f_1^* = 209.8752$.

1. Consider $N = 10$ and $p_i = 1/N$ for all $i = 1, \dots, N$. Provide the optimal value f_{10}^* of problem (2) and values for the 10 first components of the obtained solution x_{10}^*.

f_{10}^*	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}

2. Repeat the previous experiment but with probability $p = [9\ 11\ 8\ 12\ 7\ 13\ 6\ 14\ 5\ 15]/100$.

f_{10}^*	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}

3. As in item 2, but with $N = 200$ scenarios (and $p_i = 1/200$, $i = 1, \dots, 200$). Report CPU time required to solve (2). Try also $N = 2000$.

f_{200}^*	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}

[1] Use the package `mcode.sty` if you are familiar with LaTeX.

f^*_{2000}	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}

Exercise 2: the L-Shaped method

As seen in the Lecture number 10, the recourse function $\mathcal{Q}^N(x) := \sum_{i=1}^N p_i[Q(x,\xi^i)]$ can be approximated from below by optimality cuts

$$\mathcal{Q}^N(x) \geq \alpha^j + {\beta^j}^\top x\,, \quad \text{for all first-stage decision } x,$$

where $j = 1,\ldots,k$ are iteration indexes. If $x^j \notin \texttt{dom}\,\mathcal{Q}^N$, then a feasibility cut must be computed to cutoff the point x^j by the constraint $\tilde{\alpha}^j + {\tilde{\beta}^j}^\top x \leq 0$.

1. Let x^k be given. Considering problem (1), provide the formulas for computing all the coefficient cuts α^k, β^k, $\tilde{\alpha}^k$ and $\tilde{\beta}^k$.

   $\alpha^k =$
   $\beta^k =$
   $\tilde{\alpha}^k =$
   $\tilde{\beta}^k =$

2. Code a black-box that provides coefficient cuts for every given first-stage point x. Name your black-box `BBCut` and code it obeying the following structure:

   `pars = BBCut(pars)`, where `pars` is a MATLAB structure containing several fields, as described below.

   Input:

   - `pars.x` is a field containing the vector x
   - `pars.GenData` contains the name of the function that provides the problem's data (for instance, `pars.GenData = 'DataPbm'`)
   - `pars.scen` is a field containing the scenario data
   - `pars.prob` contains the probability of scenarios
   - `pars.N` is a field containing the number of scenarios.

   Output: all the input fields and

   - `pars.ind`, an indicator of the black-box performance
   - `pars.alpha` is a field containing the cut coefficient α
   - `pars.beta` is a field containing the cut coefficient β
   - `pars.Q` contains the recourse function value $Q^N(x)$

   If `pars.ind` $= 0$, then `pars.alpha` $= \alpha$ and `pars.beta` $= \beta$ are coefficients of an optimality cut. Moreover, `pars.Q` is the recourse function at x: `pars.Q` $= \mathcal{Q}^N(x)$. If `pars.ind` $=$ `j` (with $j > 0$), then `para.alpha` $= \tilde{\alpha}$ and `pars.beta` $= \tilde{\beta}$ are coefficients of a feasibility cut, and j is the index of the scenario ξ^j for which the point `pars.x` is infeasible. Finally, if `pars.ind` $= -1$, then something went wrong in the black-box. In this case, the information `pars.alpha` and `pars.beta` is meaningless.

3. Let $\mathbb{O}_k$ and $\mathbb{F}_k$ be two (disjoint) index sets contained in $\{1, 2, \ldots, k\}$. The L-Shaped master subproblem can be written as

$$\begin{cases} \min\limits_{x,r} & c^\top x + r \\ \text{s.t.} & Ax = b \\ & {\alpha^j} + {\beta^j}^\top x \le r, \qquad \forall\, j \in \mathbb{O}_k \\ & \tilde{\alpha}^j + {\tilde{\beta}^j}^\top x \le 0, \qquad \forall\, j \in \mathbb{F}_k \\ & r \in \mathbb{R},\ 0 \le x_i \le M \quad \forall\, i = 1, \ldots, n_1, \end{cases} \tag{3}$$

where $M > 0$ is an artificial constant to ensure boundedness of the feasible set (take $M = 10^5$ in your numerical experiments). Code in MATLAB the L-Shaped algorithm:

L-Shaped Algorithm

- **Step 0: initialization**. Choose a tolerance `Tol` > 0, maximum number of iterations `MaxIt` > 0, and a feasible first-stage point $x^1 \in \{x \in \mathbb{R}^{n_1}_+ : Ax = b\}$. Define $k = 1$, $f_0^{\text{up}} = \infty$ and $\mathbb{O}_0 = \mathbb{F}_0 = \emptyset$.
- **Step 1: oracle call**. Call the black-box `BBCut` at x^k.
  - If `pars.ind` $= 0$, define $\mathbb{O}_k = \mathbb{O}_{k-1} \cup \{k\}$ and $\mathbb{F}_k = \mathbb{F}_{k-1}$. If $c^\top x^k$ + `pars.Q` $< f_{k-1}^{\text{up}}$ update: $\bar{x} = x^k$ and $f_k^{\text{up}} = c^\top x^k$ + `pars.Q`. Otherwise, $f_k^{\text{up}} = f_{k-1}^{\text{up}}$.
  - If `pars.ind` > 0, define $\mathbb{O}_k = \mathbb{O}_{k-1}$ and $\mathbb{F}_k = \mathbb{F}_{k-1} \cup \{k\}$. Set $f_k^{\text{up}} = f_{k-1}^{\text{up}}$.
  - If `pars.ind` $= -1$, stop. Error in the black-box.
- **Step 2: next iterate**. Compute (x^{k+1}, r^{k+1}) by solving (3).
- **Step 3: stopping test**. Set $\Delta_k = f_k^{\text{up}} - (c^\top x^{k+1} + r^{k+1})$. If $\Delta_k \le (1 + |f_k^{\text{up}}|)$`Tol`, stop. Return $\bar{x}$ and f_k^{up}.
- **Step 4: loop**. If $k =$ `MaxIt`, stop: maximum number of iterations is reached. Return the best candidate $\bar{x}$ and its functional value f_k^{up}. Otherwise, set $k = k + 1$ and go back to Step 1.

In your experiments, define x^1 as a solution to the first-stage LP:

$$\min\ c^\top x \quad \text{s.t.} \quad Ax = b,\ \ x \ge 0\,.$$

4. Run the algorithm on the available two-stage linear programming problem. Set `Tol` $= 10^{-5}$ and report for every $N \in \{10,\ 50,\ 100,\ 200,\ 1000,\ 2000\}$ the obtained optimal value, out-of-sample value, CPU time and number of oracle calls. Consider $5,000$ out-of-sample scenarios (employ the scenarios $\tilde{\xi}$ supplied in the file `OutSample.mat`) [2].

N	# Call	CPU (s)	f_N^*	$\tilde{f}_{5000}$
10				
50				
100				
200				
1000				
2000				

5. Obtain from the set of 100,000 scenarios `scen` ten samples of $N = 200$ scenarios each. Solve the resulting ten two-stage linear programs and report the obtained optimal value f_N^* for each sample, the average and standard deviation of the ten values[3].

[2] Let x_N^* be the solution obtained by the algorithm. Then $f_N^* = c^\top x_N^* + \sum_{i=1}^N p_i Q(x_N^*, \xi^i)$ is the associate optimal value and $\tilde{f}_{5000} = c^\top x_N^* + \frac{1}{5000}\sum_{i=1}^{5000} Q(x_N^*, \tilde{\xi}^i)$ is the out-of-sample value.

[3] In order to generate randomly the samples, use the `Matlab` command: `I=randi(100000,200,1); NewSample = scen(I,:)`.

Sample	# Call	CPU (s)	f_N^*
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
Mean			
Std. Dev.			

Exercise 3: stochastic decomposition

The optimization methods considered in Exercises 1 and 2 are deterministic ones: given a set of scenarios, the optimization of the underlying problem is performed up to a given tolerance `Tol` ≥ 0. There are, however, *stochastic optimization methods* that iteratively combine optimization and sampling. Methods of this class do not consider a fixed set of scenarios and, consequently, each iteration is performed on a different approximation of the "true"stochastic program (the one formulated by considering a continuous probability distribution). A method of this class is the so-called *Stochastic Decomposition*, that at every iteration k approximates the objective function $f(x) := c^\top x + \mathcal{Q}(x)$, with $\mathcal{Q}(x) := \int_{\xi\in\Xi} Q(x,\xi)dP(\xi)$, by the easier one $f^k(x) := c^\top x + \mathcal{Q}^k(x)$, with $\mathcal{Q}^k(x) = \frac{1}{k}\sum_{i=1}^k Q(x,\xi^i)$. At iteration $k+1$, a new scenario ξ^{k+1} is generated and $f(x)$ is approximated by $f^{k+1}(x)$. It follows by the *Law of Large Numbers* that f^k converges to f when k tends to infinity. However, when k is very large the function f^k becomes hard to be evaluated. The stochastic decomposition then considers only approximations for the approximating function f^k. By assuming that the approximating errors vanish when k increases indefinitely, the stochastic decomposition will eventually be solving the "true"stochastic program, defined by function f.

Stochastic Decomposition: a bird-eye view

Assume that the random data in problem (1) is $\xi = (T, h)$. Given this assumption, the dual problem of (1b) has a feasible set that is fixed for all possible realization of the random data:

$$Q(x,\xi) = \left\{ \begin{array}{ll} \max\limits_{\pi} & (h - Tx)^\top \pi \\ \text{s.t.} & \pi \in \Pi \end{array} \right. \qquad \text{with} \quad \Pi := \{\pi : W^\top \pi \leq q\}. \tag{4}$$

Let π^k be an optimal solution of problem (4) defined at point x and scenario ξ^k. It follows by duality in linear programming that $Q(x,\xi^k) = (h^k - T^k x)^\top \pi^k$ and

$$Q(x,\xi^k) \geq (h^k - T^k x)^\top \pi \quad \forall\, \pi \in \Pi\,.$$

This inequality motivates us to save the dual solutions obtained along the iterative process in a set $\Pi^k := \{\pi^1, \pi^2, \ldots, \pi^k\} \subset \Pi$. Since Π^k is subset of Π, then

$$\max_{\pi\in\Pi^k} (h - Tx)^\top \pi \quad \leq \quad Q(x,\xi) \quad \forall\, x\,.$$

The above is a very simple optimization problem (it is essentially a matrix multiplication). This feature is exploited by the stochastic decomposition that approximates the recourse function

$$\mathcal{Q}^k(x) = \frac{1}{k}\sum_{i=1}^{k} Q(x,\xi^i) \qquad \text{by} \qquad \frac{1}{k}\sum_{i=1}^{k}(h^i - T^i x)^\top \pi^i,$$

where[4]

$$\pi^i \in \begin{cases} \arg\max_{\pi\in\Pi} \ (h^i - T^i x)^\top \pi & \text{if } i = k \\ \\ \arg\max_{\pi\in\Pi^k} \ (h^i - T^i x)^\top \pi & \text{if } i < k\,. \end{cases} \tag{5}$$

As a result, the inequality $\frac{1}{k}\sum_{i=1}^{k}(h^i - T^i x)^\top \pi^i \le \mathcal{Q}^k(x)$ holds true for all k and all first-stage point x.

Based on this development, at iteration k the stochastic decomposition draws a new scenario ξ^k and updates function $\mathcal{Q}^{k-1}$ to $\mathcal{Q}^k$. Instead of evaluating $\mathcal{Q}^k$ at a given candidate x^k, the method approximates $\mathcal{Q}^k(x^k)$ by the "much cheaper"function $\frac{1}{k}\sum_{i=1}^{k}(h^i - T^i x^k)^\top \pi^i$.

1. At iteration $k-1$, suppose that there are $k-1$ cuts approximating $\mathcal{Q}^{k-1}(x)$, i.e.,

$$\max_{j=1,\ldots,k-1}\{\alpha_j^{k-1} + {\beta_j^{k-1}}^\top x\} \le \mathcal{Q}^{k-1}(x)\,.$$

Suppose also that a new scenario k is randomly drawn, and that there exists a constant L satisfying $Q(x,\xi) \ge L$ for all x and ξ. Under this assumptions, show that

$$\mathcal{Q}^k(x) \ge \frac{k-1}{k}\mathcal{Q}^{k-1}(x) + \frac{1}{k}L \ge \max_{j=1,\ldots,k-1}\left\{\frac{k-1}{k}\alpha_j^{k-1} + \frac{k-1}{k}{\beta_j^{k-1}}^\top x\right\} + \frac{1}{k}L\,.$$

Conclude that cuts generated for $\mathcal{Q}^{k-1}$ can be easily updated to provide cuts for $\mathcal{Q}^k$.

2. Code a second black-box called `BBStoch` obeying the following structure:

[4]Notice that k LPs need to be solved in order to compute $\mathcal{Q}^k(x)$, whereas only one LP is solved to compute $\frac{1}{k}\sum_{i=1}^{k}(h^i - T^i x)^\top \pi^i$.

`pars = BBStoch(pars)`, where `pars` is as in **Exercise 2**, but with `pars.N` equal to k, the number of scenarios generated up to iteration k. The black-box must compute dual solutions π^i according to (5) in order to calculate approximate cuts. The output of the black-box is as in **Exercise 2**, but with approximate values instead. Note that a new field `pars.Pik` must be created to store the dual solutions.

3. Code the following algorithm.

Stochastic Decomposition

- **Step 0: initialization**. Let a lower bound L be given. Choose a maximum number of iterations `MaxIt` > 0, and a feasible first-stage point $x^1 \in \{x \in \mathbb{R}^{n_1}_+ : Ax = b\}$. Define $k = 1$ and $\mathbb{O}_0 = \mathbb{F}_0 = \emptyset$.
- **Step 1: oracle call**. Generate a scenario ξ^k and call the black-box `BBStoch` at x^k.
  - If `pars.ind` $= 0$, define $\mathbb{O}_k = \mathbb{O}_{k-1} \cup \{k\}$ and $\mathbb{F}_k = \mathbb{F}_{k-1}$.
  - If `pars.ind` > 0, define $\mathbb{O}_k = \mathbb{O}_{k-1}$ and $\mathbb{F}_k = \mathbb{F}_{k-1} \cup \{k\}$.
  - If `pars.ind` $= -1$, stop. Error in the black-box.
- **Step 2: cuts updating**. If $k > 1$, then set $\alpha^k_j \leftarrow \frac{k-1}{k}\alpha^{k-1}_j + \frac{1}{k}L$ and $\beta^k_j \leftarrow \frac{k-1}{k}\beta^{k-1}_j$ for all $j \in \mathbb{O}_{k-1}$.
- **Step 3: next iterate**. Compute (x^{k+1}, r^{k+1}) by solving (3) (with coefficients α^j, β^j replaced by α^k_j, β^k_j).
- **Step 4: loop**. If $k =$ `MaxIt`, stop. Call the black-box `BBCut` to compute `Q` at x^{k+1} and return x^{k+1} and $f(x^{k+1}) = c^\top x^{k+1} +$ `pars.Q`. Otherwise, set $k \leftarrow k + 1$ and go back to Step 1.

4. Run the stochastic algorithm decomposition on the available two-stage linear programming problem. Set $L = 0$ in the algorithm and report, for every `MaxIt` $\in \{10,\ 50,\ 100,\ 200,\ 1000,\ 2000\}$ the obtained optimal value, out-of-sample value, and CPU time. Consider $5,000$ out-of-sample scenarios.

`MaxIt`	# Call	CPU (s)	$f^*_{\texttt{MaxIt}}$	$\tilde{f}_{5000}$
10				
50				
100				
200				
1000				
2000				

5. Consider `MaxIt` $= 200$ and run the algorithm for ten different sample of scenarios obtained from the set of 100,000 scenarios `scen`. Report the obtained optimal value f^*_{200} for each sample, the average and standard deviation of the ten values.

Sample	# Call	CPU (s)	f^*_{2000}
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
Mean			
Std. Dev.			

Appendix: Main MATLAB file

In order to facilitate the execution of your codes, write a MATLAB code gathering the tree algorithms written by you. Name this main code `BAS`, obeying the following structure.

```
function out=BAS(N,algo)
%-------------------------------------------------------------------------
% Basic Course on Stochastic Programming, IMPA 2016
%-------------------------------------------------------------------------
% Main code for solving the Computational List of Exercises.
%
% =====
% Input:
%   N    = number of scenarios     (1 <= N <= 100000)
%   algo = algorithm to be employed to solve the problem
%          algo = 1 -> use the Equivalent Deterministic approach
%          algo = 2 -> use the L-Shaped algorithm
%          algo = 3 -> use the Stochastic Decomposition
%-------------------------------------------------------------------------
% =====
% Output:
%  out  = a Matlab structure containing at the least the following fields:
%         out.x      -> first-stage (candidate) solution
%         out.fopt   -> obtained approximation for the optimal value
%         out.cpu    -> cpu time (excluding simulation)
%         out.call   -> number of oracle calls
%         out.fsimul -> value of the function at out.x determined by the
%         out-of sample simulation
%-------------------------------------------------------------------------
%

% Loading the scenario data
load('ScenData.mat');
Nmax = size(scen,1);              % number of available scenarios
if N>Nmax
    fprintf(1,'Nmax %d \n',Nmax);
    error('N is larger than the number of available scenarios');
```

```
end
% Initializating the pars structure
pars.N       = N;            % number of considered scenarios
pars.prob    = ones(N,1)/N;  % probabilities
pars.GenData = 'DataPbm';    % function that provides the problem data
pars.scen    = scen(1:N,:);  % keeping only the first N scenarios
clear scen N;                % erase duplicate information
%-------------------------------------------------------------------------
% Calling the algorithms
if algo==1
    out = EquivDet(pars);     % calling the Equivalent deterministic solver
elseif algo==2
    out = LShaped(pars);      % calling the L-Shaped solver
elseif algo==3
    pars.MaxIt = pars.N;      % In this case, N is the maximum number of iterations
    out = StochDecomp(pars);  % calling the Stochasti Decomposition solver
else
    error('Algorithm not available');
end
%-------------------------------------------------------------------------
% Out-of-Sample simulation
out.fsimul = Simulation(out.x);
return
```