A new look at the Stochastic Dual Dynamic Programming algorithm

V. Leclère, P. Carpentier, J-P. Chancelier, A. Lenoir, F. Pacaud

SESO 2018

24/05/2018

Contents

Contents

Introduction

We are interested in multistage stochastic optimization problems of the form

$$\begin{aligned} \min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{X}_t, \boldsymbol{U}_t, \boldsymbol{\xi}_t) + K(\boldsymbol{X}_T)\Big] \\ s.t. \quad & \boldsymbol{X}_{t+1} = f_t(\boldsymbol{X}_t, \boldsymbol{U}_t, \boldsymbol{\xi}_t), \qquad \boldsymbol{X}_0 = x_0 \\ & \boldsymbol{U}_t = \pi_t(\boldsymbol{X}_t, \boldsymbol{\xi}_t) \in U_t(x, \boldsymbol{\xi}_t) \end{aligned}$$

where

- $\boldsymbol{x}_t$ is the state of the system,
- $\boldsymbol{u}_t$ is the control applied at time t,
- $\boldsymbol{\xi}_t$ is the noise happening between time t and $t+1$, assumed to be time-independent,
- π is the policy.

Time-decomposition

$$\min_{\pi} \quad \mathbb{E}\Big[L_0(x_0, \boldsymbol{U}_0, \xi_0) + \mathbb{E}\Big[\sum_{t=1}^{T-1} L_t(\boldsymbol{X}_t, \boldsymbol{U}_t, \xi_t) + K(\boldsymbol{X}_T) \Big]\Big]$$

$$s.t. \quad \boldsymbol{X}_0 = x_0$$

$$\boldsymbol{X}_1 = f_t(\boldsymbol{X}_0, \boldsymbol{U}_0, \xi_0),$$

$$\boldsymbol{U}_0 = \pi_0(x_0, \xi_0) \in U_0(x_0, \xi_0)$$

$$\boldsymbol{X}_1 = f_t(\boldsymbol{X}_0, \boldsymbol{U}_0, \xi_0)$$

$$\boldsymbol{X}_{t+1} = f_t(\boldsymbol{X}_t, \boldsymbol{U}_t, \xi_t)$$

$$\boldsymbol{U}_t = \pi_t(\boldsymbol{X}_t, \xi_t) \in U_t(\boldsymbol{X}_t, \xi_t)$$

Stochastic Dynamic Programming

By the white noise assumption, this problem can be solved by Dynamic Programming, where the Bellman functions satisfy

$$\begin{cases} V_T(x) &= K(x) \\ \hat{V}_t(x,\xi) &= \min\limits_{u_t \in U_t(x,\xi)} L_t(x,u_t,\xi) + V_{t+1} \circ \underbrace{f_t(x,u_t,\xi)}_{"x_{t+1}"} \\ V_t(x) &= \mathbb{E}\Big[\hat{V}_t(x,\boldsymbol{\xi}_t)\Big] \end{cases}$$

Indeed, π is an optimal policy if

$$\pi_t(x,\xi) \in \mathop{\arg\min}_{u_t \in U_t(x,\xi)} \Big\{ \underbrace{L_t(x,u_t,\xi)}_{\text{current cost}} + \underbrace{V_{t+1} \circ f_t(x,u_t,\xi)}_{\text{future costs}} \Big\}$$

Stochastic Dynamic Programming

By the white noise assumption, this problem can be solved by Dynamic Programming, where the Bellman functions satisfy

$$\begin{cases} V_T(x) &= K(x) \\ \hat{V}_t(x,\xi) &= \min_{u_t \in U_t(x,\xi)} L_t(x,u_t,\xi) + V_{t+1} \circ \underbrace{f_t(x,u_t,\xi)}_{"x_{t+1}"} \\ V_t(x) &= \mathbb{E}\left[\hat{V}_t(x,\boldsymbol{\xi}_t)\right] \end{cases}$$

Indeed, π is an optimal policy if

$$\pi_t(x,\xi) \in \arg\min_{u_t \in U_t(x,\xi)} \Big\{ \underbrace{L_t(x,u_t,\xi)}_{\text{current cost}} + \underbrace{V_{t+1} \circ f_t(x,u_t,\xi)}_{\text{future costs}} \Big\}$$

Bellman operator

For any time t, and any function $R : \mathbb{X} \to \mathbb{R} \cup \{+\infty\}$ we define

$$\hat{\mathcal{T}}_t(R)(x,\xi) := \min_{u_t \in \mathbb{U}} L_t(x, u_t, \xi) + R \circ f_t(x, u_t, \xi)$$

and

$$\mathcal{T}_t(R)(x) := \mathbb{E}\Big[\hat{\mathcal{T}}_t(R)(x, \boldsymbol{\xi})\Big].$$

Incidentally, R induce a policy $\pi_t^R(x,\xi)$ given by a minimizer of the above problem, and an optimal policy is given by π^V.
Finally the Bellman equation simply reads

$$\begin{cases} V_T = K \\ V_t = \mathcal{T}_t(V_{t+1}) \end{cases}$$

Bellman operator

For any time t, and any function $R : \mathbb{X} \to \mathbb{R} \cup \{+\infty\}$ we define

$$\hat{\mathcal{T}}_t(R)(x,\xi) := \min_{u_t \in \mathbb{U}} L_t(x, u_t, \xi) + R \circ f_t(x, u_t, \xi)$$

and

$$\mathcal{T}_t(R)(x) := \mathbb{E}\Big[\hat{\mathcal{T}}_t(R)(x, \boldsymbol{\xi})\Big].$$

Incidentally, R induce a policy $\pi_t^R(x,\xi)$ given by a minimizer of the above problem, and an optimal policy is given by π^V.

Finally the Bellman equation simply reads

$$\begin{cases} V_T = K \\ V_t = \mathcal{T}_t(V_{t+1}) \end{cases}$$

Bellman operator

For any time t, and any function $R : \mathbb{X} \to \mathbb{R} \cup \{+\infty\}$ we define

$$\hat{\mathcal{T}}_t(R)(x, \xi) := \min_{u_t \in \mathbb{U}} L_t(x, u_t, \xi) + R \circ f_t(x, u_t, \xi)$$

and

$$\mathcal{T}_t(R)(x) := \mathbb{E}\Big[\hat{\mathcal{T}}_t(R)(x, \boldsymbol{\xi})\Big].$$

Incidentally, R induce a policy $\pi_t^R(x, \xi)$ given by a minimizer of the above problem, and an optimal policy is given by π^V.
Finally the Bellman equation simply reads

$$\begin{cases} V_T = K \\ V_t = \mathcal{T}_t(V_{t+1}) \end{cases}$$

Properties of the Bellman operator

- Monotonicity:

$$V \leq \overline{V} \quad \Rightarrow \mathcal{T}_t\big(V\big) \leq \mathcal{T}_t\big(\overline{V}\big)$$

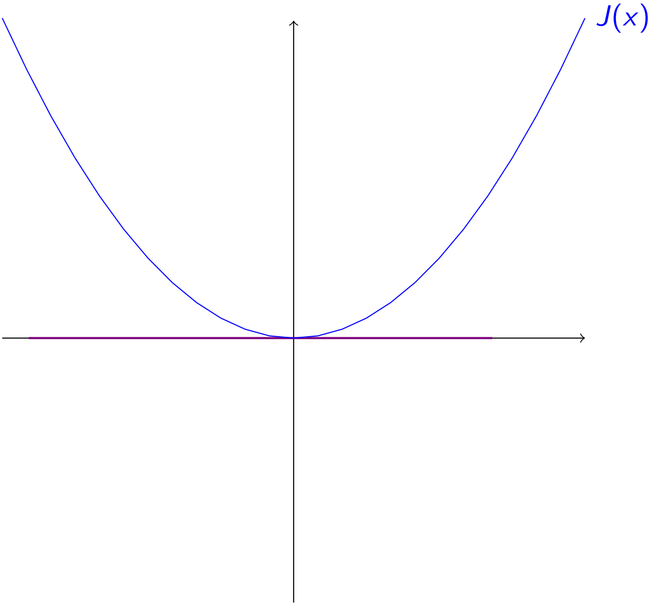
- Convexity: if L_t is jointly convex in (x, u), V is convex, and f_t is affine then

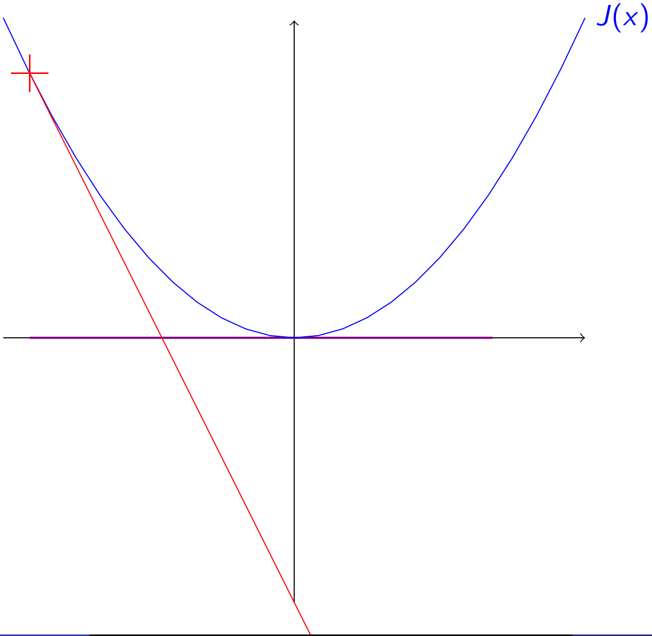
$$x \mapsto \mathcal{T}_t\big(V\big)(x) \quad \text{is convex}$$

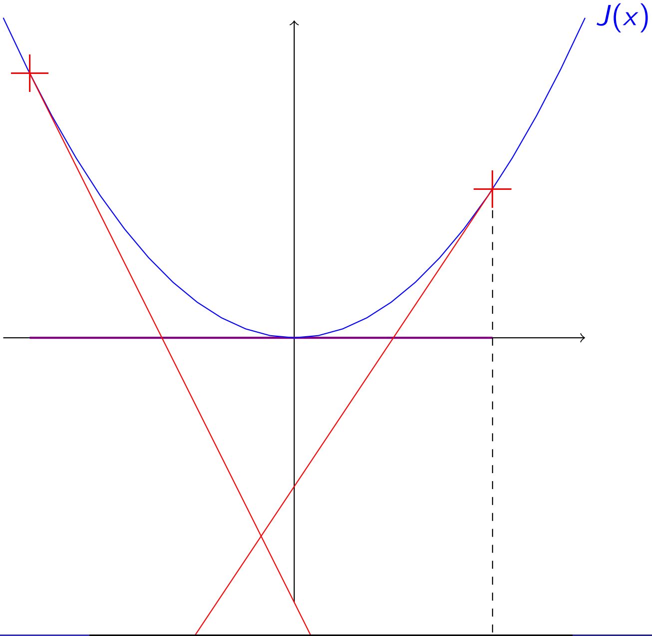
- Polyhedrality: for any polyhedral function V, if L_t is also polyhedral, and f_t affine, then

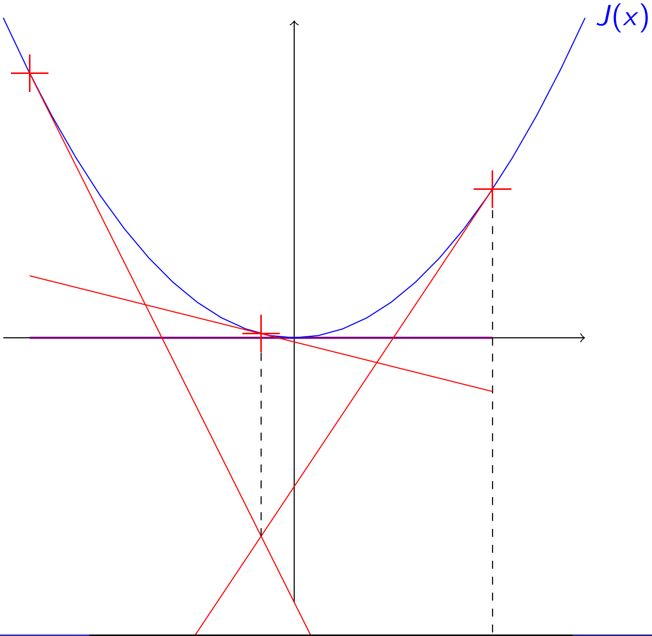
$$x \mapsto \mathcal{T}_t\big(V\big)(x) \quad \text{is polyhedral}$$

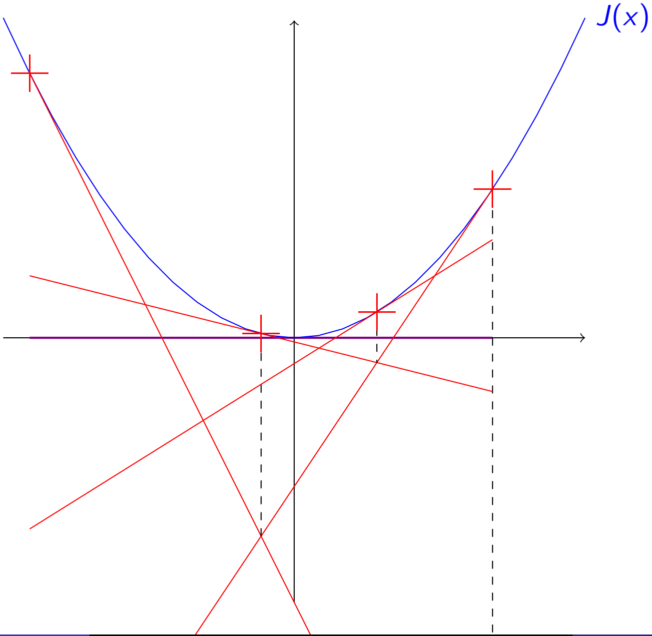
Contents


$J(x)$


$J(x)$


$J(x)$


$J(x)$


J(x)

Duality property

- Consider $J : \mathbb{X} \times \mathbb{U} \to \mathbb{R}$ jointly convex, and define

$$\varphi(x) = \min_{u \in \mathbb{U}} J(x, u)$$

- Then we can obtain a subgradient $\lambda \in \partial\varphi(x_0)$ as the dual multiplier of

$$\begin{aligned} \min_{x,u} \quad & J(x, u), \\ s.t. \quad & x_0 - x = 0 \qquad\qquad [\lambda] \end{aligned}$$

  (This is the marginal interpretation of the multiplier)
- In particular, we have that

$$\varphi(\cdot) \geq \varphi(x_0) + \langle \lambda, \cdot - x_0 \rangle$$

Computing cuts (1/2)

Suppose that we have $\underline{V}_{t+1}^{(k+1)} \leq V_{t+1}$

$$\begin{aligned} \hat{\beta}_t^{(k+1)} &= \min_{x,u} \quad L_t(x, u) + \underline{V}_{t+1}^{(k+1)} \circ f_t(x, u) \\ & s.t \quad x = x_t^{(k)} \qquad [\hat{\lambda}_t^{(k+1)}] \end{aligned}$$

This can also be written as

$$\begin{aligned} \hat{\beta}_t^{(k+1)} &= \hat{\mathcal{T}}_t\left(\underline{V}_{t+1}^{(k+1)}\right)(x) \\ \hat{\lambda}_t^{(k+1)} &\in \partial_x \hat{\mathcal{T}}_t\left(\underline{V}_{t+1}^{(k+1)}\right)(x) \end{aligned}$$

Thus, $\hat{\mathcal{C}}_t^{(k+1)} : x \mapsto \hat{\beta}_t^{(k+1)} + \left\langle \hat{\lambda}_t^{(k+1)}, x - x_t^{(k)} \right\rangle$ satisfy

$$\hat{\mathcal{C}}_t^{(k+1)}(x) \leq \hat{\mathcal{T}}_t\left(\underline{V}_{t+1}^{(k+1)}\right)(x) \leq \hat{\mathcal{T}}_t\left(V_{t+1}\right)(x) = \hat{V}_t(x)$$

Computing cuts (1/2)

Suppose that we have $\underline{V}_{t+1}^{(k+1)} \leq V_{t+1}$

$$\hat{\beta}_t^{(k+1)} = \min_{x,u} \quad L_t(x,u) + \underline{V}_{t+1}^{(k+1)} \circ f_t(x,u)$$
$$s.t \quad x = x_t^{(k)} \qquad [\hat{\lambda}_t^{(k+1)}]$$

This can also be written as

$$\hat{\beta}_t^{(k+1)} = \hat{\mathcal{T}}_t\left(\underline{V}_{t+1}^{(k+1)}\right)(x)$$
$$\hat{\lambda}_t^{(k+1)} \in \partial_x \hat{\mathcal{T}}_t\left(\underline{V}_{t+1}^{(k+1)}\right)(x)$$

Thus, $\hat{\mathcal{C}}_t^{(k+1)} : x \mapsto \hat{\beta}_t^{(k+1)} + \left\langle \hat{\lambda}_t^{(k+1)}, x - x_t^{(k)} \right\rangle$ satisfy

$$\hat{\mathcal{C}}_t^{(k+1)}(x) \leq \hat{\mathcal{T}}_t\left(\underline{V}_{t+1}^{(k+1)}\right)(x) \leq \hat{\mathcal{T}}_t\left(V_{t+1}\right)(x) = \hat{V}_t(x)$$

Computing cuts (1/2)

Suppose that we have $\underline{V}_{t+1}^{(k+1)} \leq V_{t+1}$

$$\hat{\beta}_t^{(k+1)} = \min_{x,u} \quad L_t(x, u \quad + \underline{V}_{t+1}^{(k+1)} \circ f_t(x, u \quad)$$
$$s.t \quad x = x_t^{(k)} \qquad [\hat{\lambda}_t^{(k+1)} \quad]$$

This can also be written as

$$\hat{\beta}_t^{(k+1)} = \hat{\mathcal{T}}_t \left(\underline{V}_{t+1}^{(k+1)} \right) (x \quad)$$
$$\hat{\lambda}_t^{(k+1)} \in \partial_x \hat{\mathcal{T}}_t \left(\underline{V}_{t+1}^{(k+1)} \right) (x \quad)$$

Thus, $\hat{\mathcal{C}}_t^{(k+1)} \quad : x \mapsto \hat{\beta}_t^{(k+1)} \quad + \left\langle \hat{\lambda}_t^{(k+1)} \quad , x - x_t^{(k)} \right\rangle$ satisfy

$$\hat{\mathcal{C}}_t^{(k+1)} \ (x) \leq \hat{\mathcal{T}}_t \left(\underline{V}_{t+1}^{(k+1)} \right) (x \quad) \leq \hat{\mathcal{T}}_t \left(V_{t+1} \right) (x \quad) = \hat{V}_t(x \quad)$$

Computing cuts (1/2)

Suppose that we have $\underline{V}_{t+1}^{(k+1)} \leq V_{t+1}$

$$\hat{\beta}_t^{(k+1)}(\xi) = \min_{x,u} \quad L_t(x,u,\xi) + \underline{V}_{t+1}^{(k+1)} \circ f_t(x,u,\xi)$$
$$s.t \quad x = x_t^{(k)} \qquad [\hat{\lambda}_t^{(k+1)}(\xi)]$$

This can also be written as

$$\hat{\beta}_t^{(k+1)}(\xi) = \hat{\mathcal{T}}_t\left(\underline{V}_{t+1}^{(k+1)}\right)(x,\xi)$$
$$\hat{\lambda}_t^{(k+1)}(\xi) \in \partial_x \hat{\mathcal{T}}_t\left(\underline{V}_{t+1}^{(k+1)}\right)(x,\xi)$$

Thus, $\hat{\mathcal{C}}_t^{(k+1),\xi} : x \mapsto \hat{\beta}_t^{(k+1)}(\xi) + \left\langle \hat{\lambda}_t^{(k+1)}(\xi), x - x_t^{(k)} \right\rangle$ satisfy, for all ξ,

$$\hat{\mathcal{C}}_t^{(k+1),\xi}(x) \leq \hat{\mathcal{T}}_t\left(\underline{V}_{t+1}^{(k+1)}\right)(x,\xi) \leq \hat{\mathcal{T}}_t\left(V_{t+1}\right)(x,\xi) = \hat{V}_t(x,\xi)$$

Computing cuts (2/2)

Thus,

$$\hat{\beta}_t^{(k+1)}(\xi) + \left\langle \hat{\lambda}_t^{(k+1)}(\xi), x - x_t^{(k)} \right\rangle \leq \hat{V}_t(x, \xi)$$

for each realization ξ of $\boldsymbol{\xi}_t$.
Replacing ξ by $\boldsymbol{\xi}_t$ and taking the expectation yields

$$\mathbb{E}\left[\hat{\beta}_t^{(k+1)}(\boldsymbol{\xi}_t)\right] + \mathbb{E}\left[\left\langle \hat{\lambda}_t^{(k+1)}(\boldsymbol{\xi}_t), x - x_t^{(k)} \right\rangle\right] \leq \mathbb{E}\left[\hat{V}_t(x, \boldsymbol{\xi}_t)\right] = V_t(x)$$

and finally we have the cut

$$\beta_t^{(k+1)} + \left\langle \lambda_t^{(k+1)}, \cdot - x_t^{(k)} \right\rangle \leq V_t$$

where

$$\begin{cases} \beta_t^{(k+1)} & := & \mathbb{E}\left[\hat{\beta}_t^{(k+1)}(\boldsymbol{\xi}_t)\right] = \mathcal{T}_t\left(\underline{V}_{t+1}^{(k)}\right)(x) \\ \lambda_t^{(k+1)} & := & \mathbb{E}\left[\hat{\lambda}_t^{(k+1)}(\boldsymbol{\xi}_t)\right] \in \partial \mathcal{T}_t\left(\underline{V}_{t+1}^{(k)}\right)(x) \end{cases}$$

Computing cuts (2/2)

Thus,

$$\hat{\beta}_t^{(k+1)}(\xi) + \left\langle \hat{\lambda}_t^{(k+1)}(\xi), x - x_t^{(k)} \right\rangle \leq \hat{V}_t(x, \xi)$$

for each realization ξ of $\boldsymbol{\xi}_t$.
Replacing ξ by $\boldsymbol{\xi}_t$ and taking the expectation yields

$$\mathbb{E}\left[\hat{\beta}_t^{(k+1)}(\boldsymbol{\xi}_t)\right] + \mathbb{E}\left[\left\langle \hat{\lambda}_t^{(k+1)}(\boldsymbol{\xi}_t), x - x_t^{(k)} \right\rangle\right] \leq \mathbb{E}\left[\hat{V}_t(x, \boldsymbol{\xi}_t)\right] = V_t(x)$$

and finally we have the cut

$$\beta_t^{(k+1)} + \left\langle \lambda_t^{(k+1)}, \cdot - x_t^{(k)} \right\rangle \leq V_t$$

where

$$\begin{cases} \beta_t^{(k+1)} & := & \mathbb{E}\left[\hat{\beta}_t^{(k+1)}(\boldsymbol{\xi}_t)\right] = \mathcal{T}_t\Big(\underline{V}_{t+1}^{(k)}\Big)(x) \\ \lambda_t^{(k+1)} & := & \mathbb{E}\left[\hat{\lambda}_t^{(k+1)}(\boldsymbol{\xi}_t)\right] \in \partial\mathcal{T}_t\big(\underline{V}_{t+1}^{(k)}\big)(x) \end{cases}$$

Contents

SDDP algorithm

Under linear dynamics, and convex costs, the SDDP algorithm iteratively constructs polyhedral outer approximations of V_t.

More precisely, at iteration k

- We have polyhedral functions $\underline{V}_t^k(\cdot) = \max_{\kappa \leq k} \left\{ \left\langle \lambda_t^\kappa , \cdot \right\rangle + \beta_t^\kappa \right\}$, such that $\underline{V}_t^k \leq V_t$.
- **Forward pass**: We simulate the dynamical system, along one scenario, according to $\pi^{\underline{V}^k}$, yielding a trajectory $\{\underline{x}_t^k\}_{t \in [\![0,T]\!]}$.
- **Backward pass**: We compute cuts

$$\mathcal{C}_t^k : x \mapsto \left\langle \lambda_t^{k+1} , x \right\rangle + \beta_t^{k+1} \leq V_t$$

along this trajectory, and update our outer approximations.

SDDP strengths

- SDDP is a widely used algorithm in the energy community, with multiple applications in
  - mid and long term water storage management problem,
  - long-term investment problems,
  - ...
- Recent works have presented extensions of the algorithm to
  - deal with some non-convexity,
  - treat risk-averse or distributionally robust problems,
  - incorporate integer variables.
- Multiple numerical improvements have been proposed
  - cut selection
  - regularization
  - multi-cut or ε-resolution

SDDP weaknesses

There are still some gaps in our knowledge of this approach:

- there is no convergence speed guaranteed,
- regularization methods are not mature yet,
- there is no good stopping test.

SDDP Stopping test

- Exact lower bound of the problem : $\underline{V}_0^k(x_0)$.
- Upper-bound estimated by Monte-Carlo simulation yielding costly statistical stopping tests (Pereira Pinto (1991) or Shapiro (2011))
- Alternative statistical tests have been proposed (see Homem de Mello et al (2011))
- Exact upper-bound computation has been proposed by Philpott et al (2013) but without any proof of convergence, leading to possibly not converging stopping tests.

Contents

Linear Bellman Operator

An operator $\mathcal{B} : F(\mathbb{R}^{n_x}) \to F(\mathbb{R}^{n_x})$ is said to be a *linear Bellman operator* (LBO) if it is defined as follows

$$\mathcal{B}(R) : x \mapsto \inf_{(\boldsymbol{u},\boldsymbol{y})} \mathbb{E}\Big[\boldsymbol{c}^\top \boldsymbol{u} + R(\boldsymbol{y})\Big]$$

$$s.t. \quad Tx + \mathcal{W}_u(\boldsymbol{u}) + \mathcal{W}_y(\boldsymbol{y}) \leq \boldsymbol{h}$$

where $\mathcal{W}_u : \mathcal{L}^0(\mathbb{R}^{n_u}) \to \mathcal{L}^0(\mathbb{R}^{n_c})$ and $\mathcal{W}_y : \mathcal{L}^0(\mathbb{R}^{n_x}) \to \mathcal{L}^0(\mathbb{R}^{n_c})$ are two linear operators. We denote $S(R)(x)$ the set of $\boldsymbol{y}$ that are part of optimal solutions to the above problem.
We also define $\mathcal{G}(x)$

$$\mathcal{G}(x) := \big\{(\boldsymbol{u},\boldsymbol{y}) \mid Tx + \mathcal{W}_u(\boldsymbol{u}) + \mathcal{W}_y(\boldsymbol{y}) \leq \boldsymbol{h}\big\}$$

Examples

- Linear point-wise operator:

$$\begin{array}{rccc} \mathcal{W} & : & \mathcal{L}^0(\mathbb{R}^{n_x}) & \to & \mathcal{L}^0(\mathbb{R}^{n_c}) \\ & & (\omega \mapsto \boldsymbol{y}(\omega)) & \mapsto & (\omega \mapsto A\boldsymbol{y}(\omega)) \end{array}$$

  Such an operator allows to encode almost sure constraints.

- Linear expected operator:

$$\begin{array}{rccc} \mathcal{W} & : & \mathcal{L}^0(\mathbb{R}^{n_x}) & \to & \mathcal{L}^0(\mathbb{R}^{n_c}) \\ & & (\omega \mapsto \boldsymbol{y}(\omega)) & \mapsto & (\omega \mapsto A\,\mathbb{E}(\boldsymbol{y})) \end{array}$$

  Such an operator allows to encode constraints in expectation.

Relatively Complete Recourse and cuts

Definition (Relatively Complete Recourse)

We say that the pair $(\mathcal{B}, R)$ satisfy a *relatively complete recourse* (RCR) assumption if for all $x \in \mathrm{dom}(\mathcal{G})$ there exists admissible controls $(\boldsymbol{u}, \boldsymbol{y}) \in \mathcal{G}(x)$ such that $\boldsymbol{y} \in \mathrm{dom}(R)$.

Cut

If R is proper and polyhedral, with RCR assumption, then $\mathcal{B}(R)$ is a proper polyhedral function.

Furthermore, computing $\mathcal{B}(R)(x)$ consists of solving a linear problem which also generates a supporting hyperplane of $\mathcal{B}(R)$, that is, a pair $(\lambda, \beta) \in \mathbb{R}^{n_x} \times \mathbb{R}$ such that

$$\begin{cases} \langle \lambda \,, \cdot \rangle + \beta \leq \mathcal{B}(R)(\cdot) \\ \langle \lambda \,, x \rangle + \beta = \mathcal{B}(R)(x) \end{cases}$$

Contents

Setting

Consider a *compatible* sequence of LBO $\{\mathcal{B}_t\}_{t \in [\![0, T-1]\!]}$, that is, such that all admissible controls of $\mathcal{B}_t$ lead to admissible states of $\mathcal{B}_{t+1}$.

Consider a sequence of functions such that

$$\begin{cases} R_T = K \\ R_t = \mathcal{B}_t(R_{t+1}) \quad \forall t \in [\![0, T-1]\!] \end{cases}$$

Then, the abstract SDDP algorithm generates a sequence of lower polyhedral approximations of R_t. In a forward pass it simulates a trajectory of states, along which the approximation is refined in the backward pass.

Setting

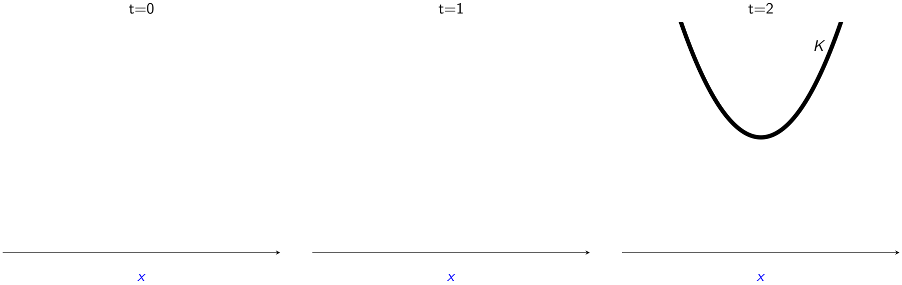
Consider a *compatible* sequence of LBO $\{\mathcal{B}_t\}_{t \in [\![0, T-1]\!]}$, that is, such that all admissible controls of $\mathcal{B}_t$ lead to admissible states of $\mathcal{B}_{t+1}$.

Consider a sequence of functions such that

$$\begin{cases} R_T = K \\ R_t = \mathcal{B}_t(R_{t+1}) \quad \forall t \in [\![0, T-1]\!] \end{cases}$$

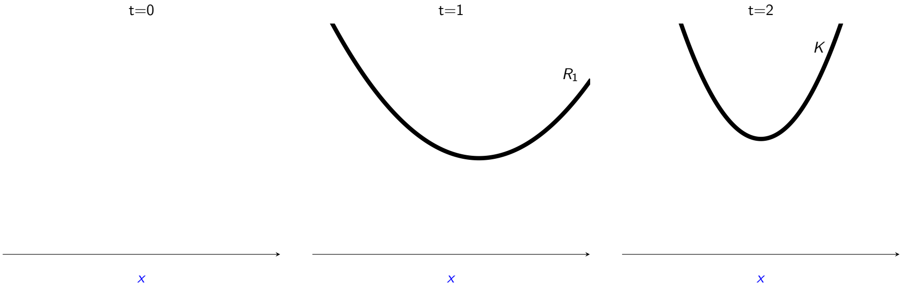
Then, the abstract SDDP algorithm generates a sequence of lower polyhedral approximations of R_t. In a forward pass it simulates a trajectory of states, along which the approximation is refined in the backward pass.

Abstract SDDP



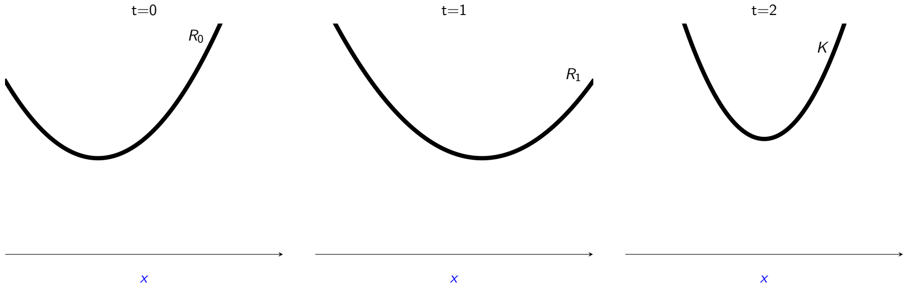
Final Cost $V_2 = K$

Abstract SDDP



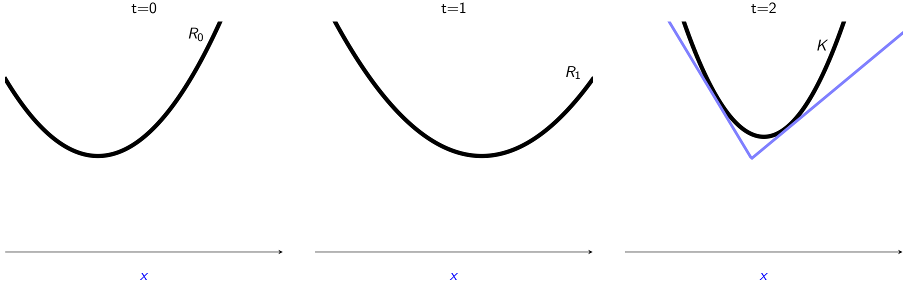
Real Bellman function $V_1 = T_1(V_2)$

Abstract SDDP



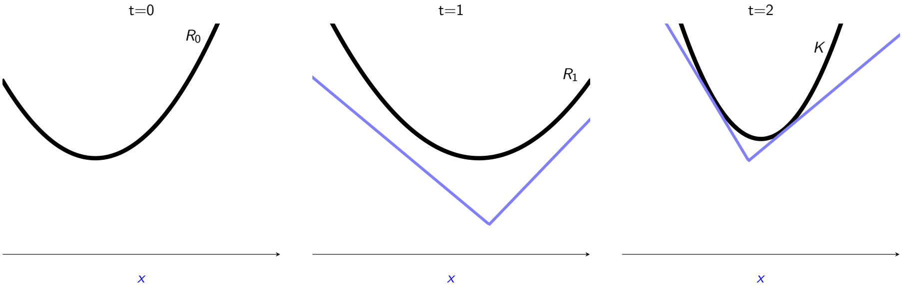
Real Bellman function $V_0 = T_0(V_1)$

Abstract SDDP



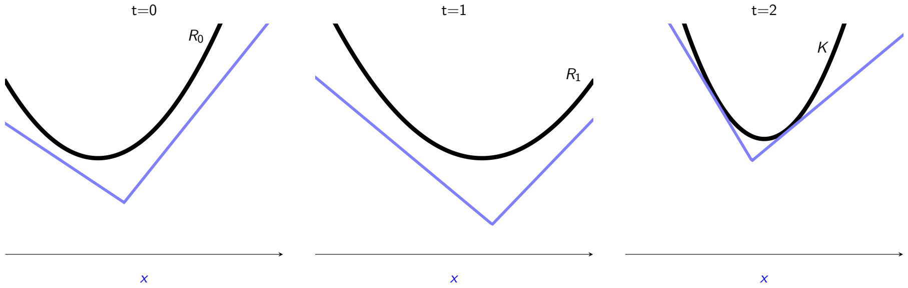
Lower polyhedral approximation $\underline{K}$ of K

Abstract SDDP



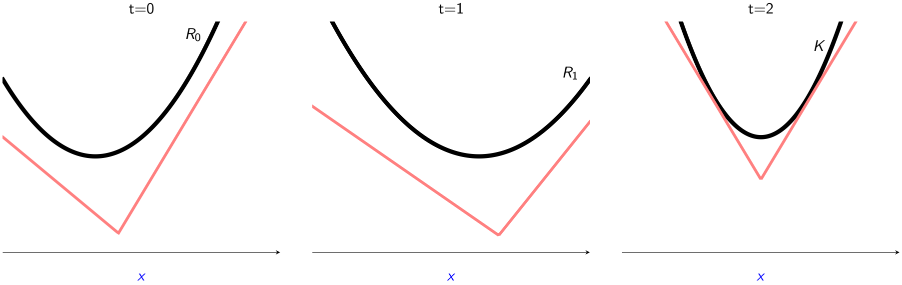
Lower polyhedral approximation $\underline{V}_1 = T_t(\underline{K})$ of V_1

Abstract SDDP



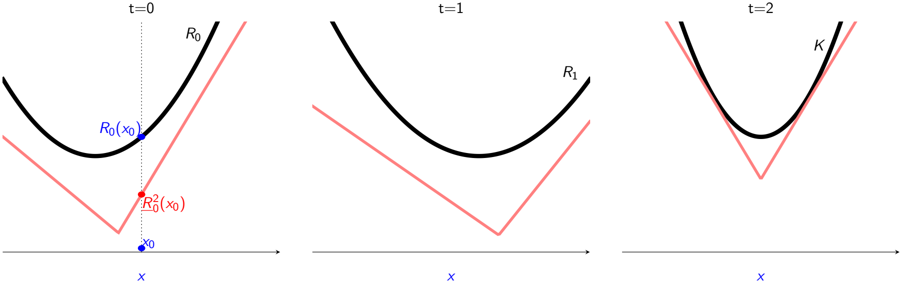
Lower polyhedral approximation $\underline{V}_0 = \mathcal{T}_t(\underline{V}_1)$ of V_0

Abstract SDDP



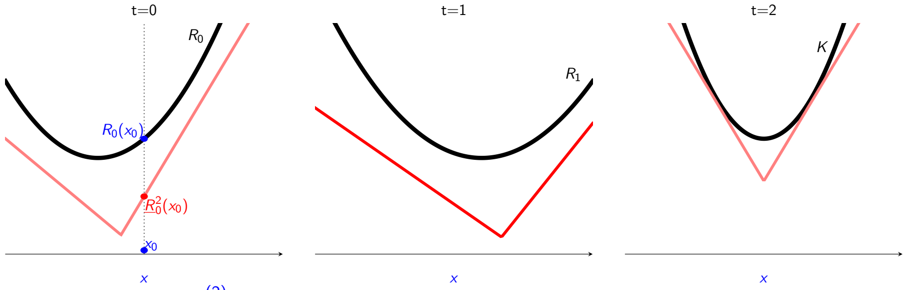
Assume that we have lower polyhedral approximations of V_t

Abstract SDDP



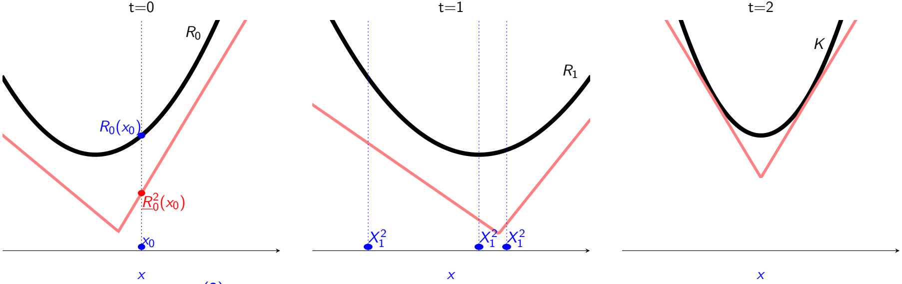
Thus we have a lower bound on the value of our problem

Abstract SDDP



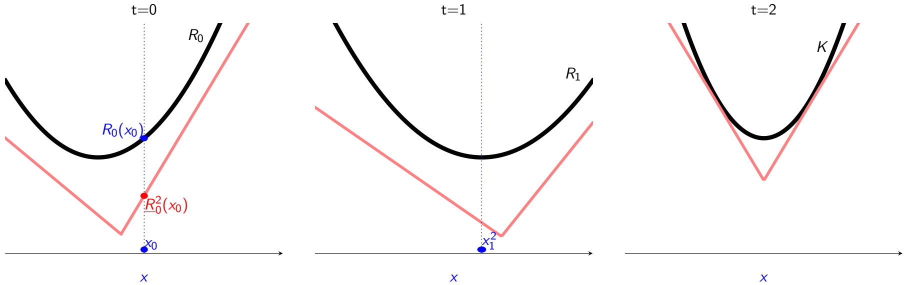
We apply $\pi_0^{\underline{V}_1^{(2)}}$ to x_0 and obtain $\boldsymbol{X}_1^{(2)}$

Abstract SDDP



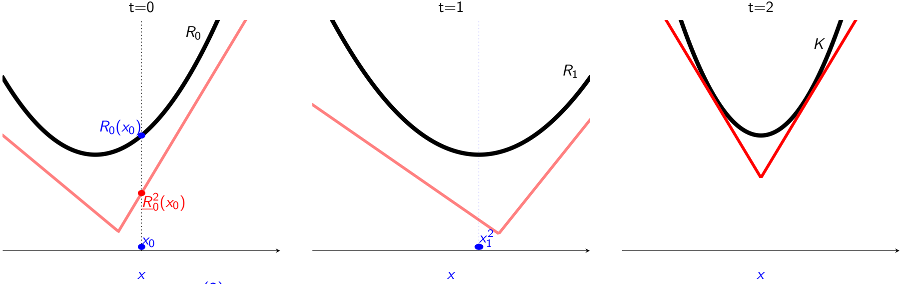
We apply $\pi_0^{\underline{V}_1^{(2)}}$ to x_0 and obtain $\boldsymbol{X}_1^{(2)}$

Abstract SDDP



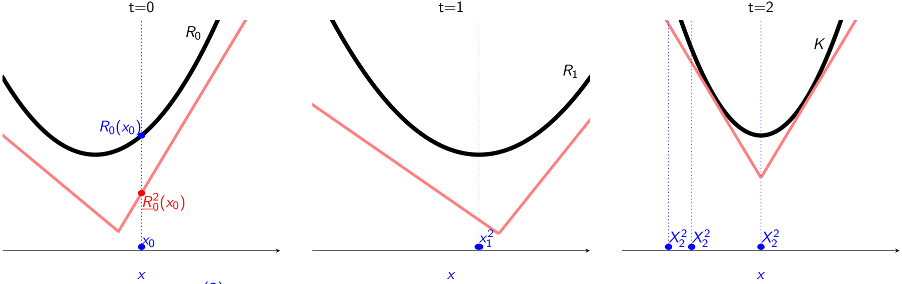
Draw a random realisation $x_1^{(2)}$ of $\boldsymbol{X}_1^{(2)}$

Abstract SDDP



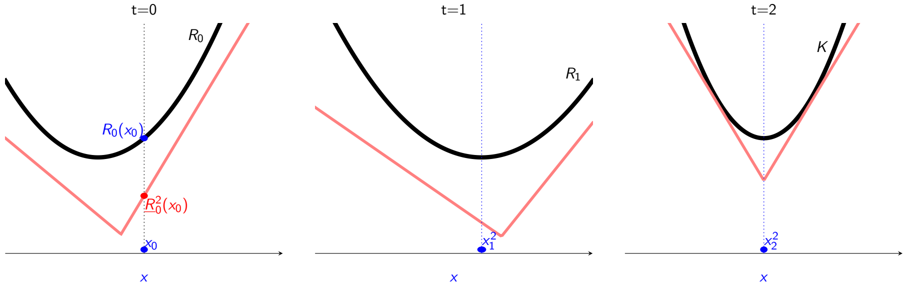
We apply $\pi_1^{\underline{V}_1^{(2)}}$ to $x_1^{(2)}$ and obtain $\boldsymbol{X}_2^{(2)}$

Abstract SDDP



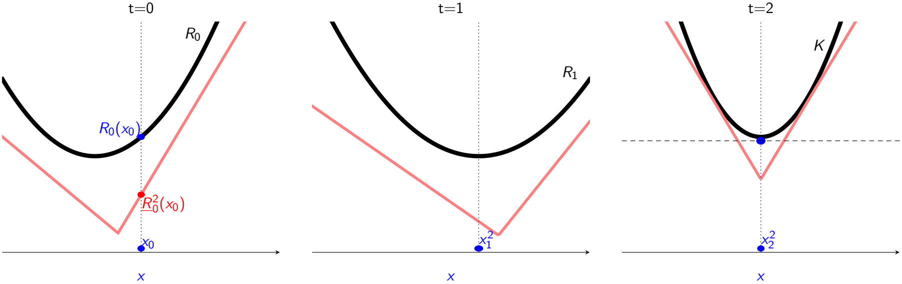
We apply $\pi_1^{\underline{V}_1^{(2)}}$ to $x_1^{(2)}$ and obtain $\boldsymbol{X}_2^{(2)}$

Abstract SDDP



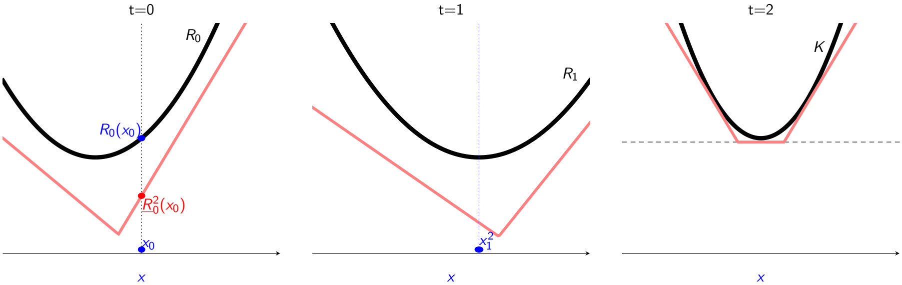
Draw a random realisation $x_2^{(2)}$ of $\boldsymbol{X}_2^{(2)}$

Abstract SDDP



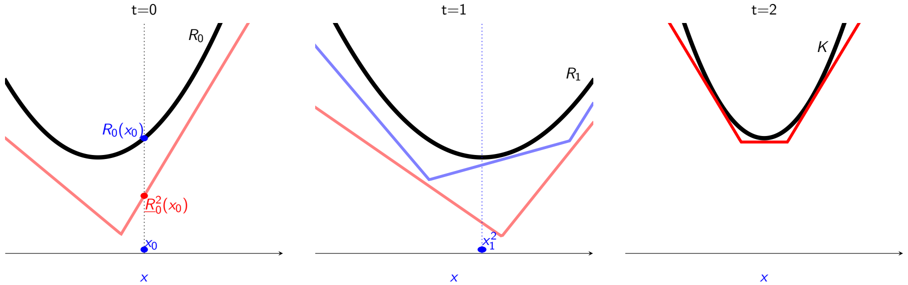
Compute a cut for K at $x_2^{(2)}$

Abstract SDDP



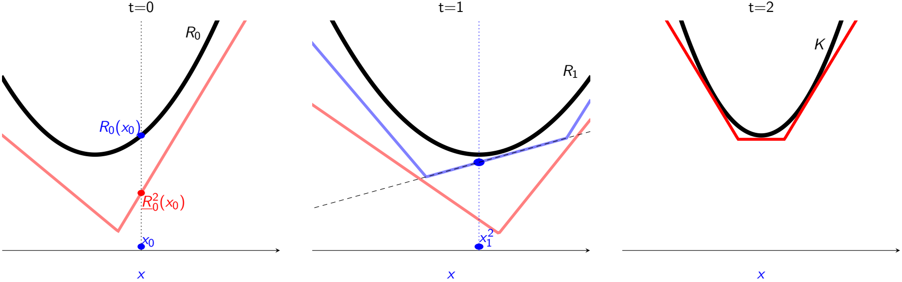
Add the cut to $\underline{V}_2^{(2)}$ which gives $\underline{V}_2^{(3)}$

Abstract SDDP



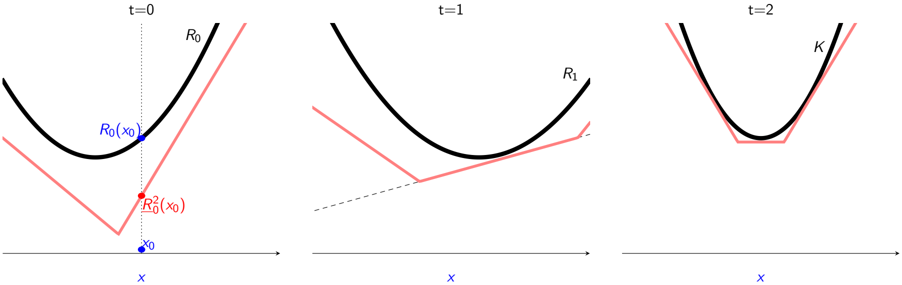
A new lower approximation of V_1 is $T_1(\underline{V}_2^{(3)})$

Abstract SDDP



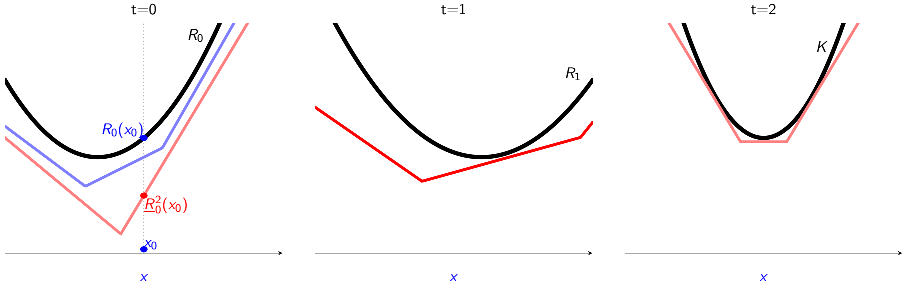
We only compute the face active at $x_1^{(2)}$

Abstract SDDP



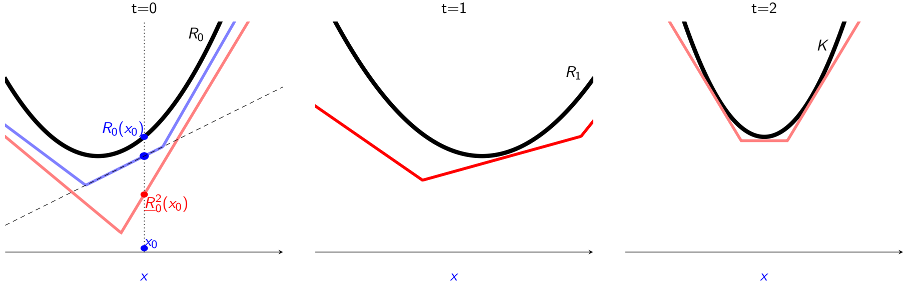
Add the cut to $\underline{V}_1^{(2)}$ which gives $\underline{V}_1^{(3)}$

Abstract SDDP



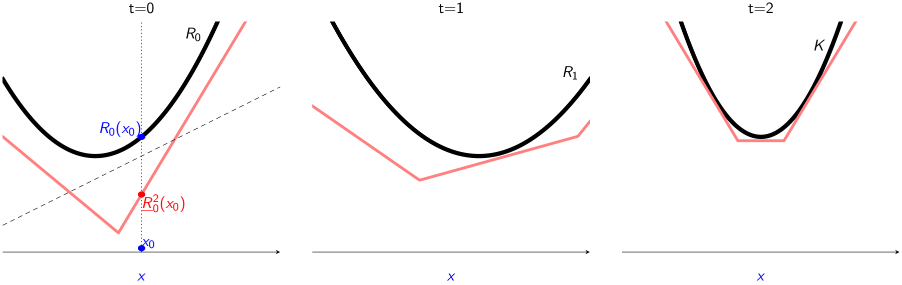
A new lower approximation of V_0 is $T_0(\underline{V}_1^{(3)})$

Abstract SDDP



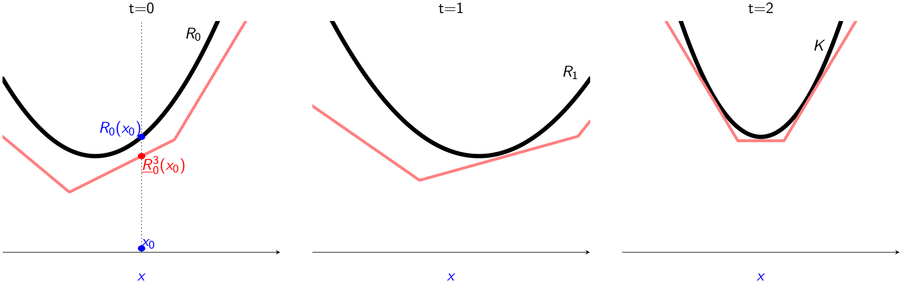
We only compute the face active at x_0

Abstract SDDP



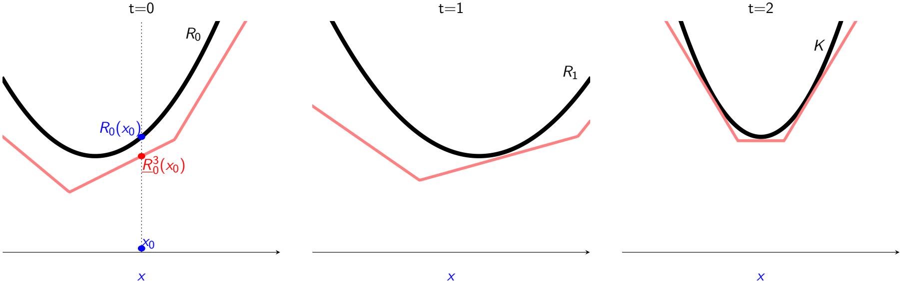
We only compute the face active at x_0

Abstract SDDP



We obtain a new lower bound

Abstract SDDP



We obtain a new lower bound

Data: Initial point x_0
$\underline{R}_t^{(0)} \leftarrow -\infty$
for $k : 0,\ 1, \dots$ **do**
// Forward Pass : compute a set of trial points $\{x_t^k\}_{t\in[\![0,T]\!]}$
$x_0^k \leftarrow x_0$
for $t : 0$ **to** $T-1$ **do**
select $\boldsymbol{x}_{t+1}^k \in \arg\min \mathcal{T}_t(\underline{R}_{t+1}^k)(x_t^k)$
Randomly select $\omega_t^k \in \Omega$
$x_{t+1}^k \leftarrow \boldsymbol{x}_{t+1}^k(\omega_t^k)$
end for
// Backward Pass : refine the lower approximations at the trial points
$\underline{R}_T^{k+1} \leftarrow K$
for $t : T-1$ **to** 0 **do**
$\theta_t^{k+1} \leftarrow \mathcal{B}_t(\underline{R}_{t+1}^{k+1})(x_t^k)$ // cut coefficients
select $\lambda_t^{k+1} \in \partial\mathcal{B}_t(\underline{R}_{t+1}^{k+1})(x_t^k)$
$\beta_t^{k+1} \leftarrow \theta_t^{k+1} - \langle \lambda_t^{k+1}, \overline{x}_t^k \rangle$
Define $\mathcal{C}_t^{k+1} : x \mapsto \langle \lambda_t^{k+1}, x \rangle + \beta_t^{k+1}$ // new cut
$\underline{R}_t^{k+1} \leftarrow \max\{\underline{R}_t^k, \mathcal{C}_t^{k+1}\}$ // update lower approximation
end for
STOP if some stopping test is satisfied
end for

Absract SDDP convergence

Theorem

Assume that Ω is finite, $R(x_0)$ is finite, and $\{\mathcal{B}_t\}_t$ is compatible. Further assume that, for all $t \in [\![0, T]\!]$ there exists compact sets X_t such that, for all k, $x_t^k \in X_t$ (e.g. $\mathcal{B}_t$ have compact domain).

Then, $(\underline{R}_t^k)_{k\in\mathbb{N}}$ is a non-decreasing sequence of lower approximations of R_t, and $\lim_k \underline{R}_0^k(x_0) = R_0(x_0)$, for $t \in [\![0, T-1]\!]$.

Further, the cuts coefficients generated remain in a compact set.

Contents

Stochastic Optimization problem

Recall the optimization problem

$$\begin{aligned} \min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{X}_t, \boldsymbol{U}_t, \boldsymbol{\xi}_t) + K(\boldsymbol{X}_T)\Big] \\ s.t. \quad & \boldsymbol{X}_{t+1} = f_t(\boldsymbol{X}_t, \boldsymbol{U}_t, \boldsymbol{\xi}_t) \\ & \boldsymbol{U}_t = \pi_t(\boldsymbol{X}_t, \boldsymbol{\xi}_t) \in U_t(x, \xi) \end{aligned}$$

With associated Dynamic Programming equation

$$\begin{cases} V_T(x) & = \quad K(x) \\ \hat{V}_t(x, \xi) & = \quad \min_{u_t \in U_t(x,\xi)} L_t(x, u_t, \xi) + V_{t+1} \circ f_t(x, u_t, \xi) \\ V_t(x) & = \quad \mathbb{E}\Big[\hat{V}_t(x, \boldsymbol{\xi}_t)\Big] \end{cases}$$

Primal Bellman equation

Consequently we introduce the following Bellman operators

$$\hat{\mathcal{T}}_t(R)(x,\xi) = \min_{u_t \in \mathbb{U}} L_t(x,u_t,\xi) + R \circ f_t(x,u_t,\xi)$$

and

$$\mathcal{T}_t(R) : x \mapsto \mathbb{E}\big[\hat{\mathcal{T}}_t(R(x,\boldsymbol{\xi}_t))\big]$$

Which allow to rewrite the Dynamic Programming equation as

$$\begin{cases} V_T &= K \\ \hat{V}_t &= \hat{\mathcal{T}}_t(V_{t+1}) \\ V_t &= \mathcal{T}_t(V_{t+1}) \end{cases}$$

Contents

What are the specificities of Primal SDDP algorithm ?

- In the forward pass you don't need to solve $\mathcal{T}_t(\underline{V}_{t+1}^k)(x_t^k)$
- It is enough to solve $\hat{\mathcal{T}}_t(\underline{V}_{t+1}^k)(x_t^k, \xi_t^k)$
- In the backward pass you need to solve $\hat{\mathcal{T}}_t(\underline{V}_{t+1}^{k+1})(x_t^k, \xi)$ for all $\xi \in \text{supp}(\boldsymbol{\xi}_t)$
- And the cut coefficients are computed as the mean of the cut coefficient associated to $\hat{V}_t$
- And that's it !

What are the specificities of Primal SDDP algorithm ?

- In the forward pass you don't need to solve $\mathcal{T}_t(\underline{V}_{t+1}^k)(x_t^k)$
- It is enough to solve $\hat{\mathcal{T}}_t(\underline{V}_{t+1}^k)(x_t^k, \xi_t^k)$
- In the backward pass you need to solve $\hat{\mathcal{T}}_t(\underline{V}_{t+1}^{k+1})(x_t^k, \xi)$ for all $\xi \in \mathrm{supp}(\boldsymbol{\xi}_t)$
- And the cut coefficients are computed as the mean of the cut coefficient associated to $\hat{V}_t$
- And that's it !

What are the specificities of Primal SDDP algorithm ?

- In the forward pass you don't need to solve $\mathcal{T}_t(\underline{V}_{t+1}^k)(x_t^k)$
- It is enough to solve $\hat{\mathcal{T}}_t(\underline{V}_{t+1}^k)(x_t^k, \xi_t^k)$
- In the backward pass you need to solve $\hat{\mathcal{T}}_t(\underline{V}_{t+1}^{k+1})(x_t^k, \xi)$ for all $\xi \in \mathrm{supp}(\boldsymbol{\xi}_t)$
- And the cut coefficients are computed as the mean of the cut coefficient associated to $\hat{V}_t$
- And that's it !

Data: initial point x_0, initial lower bounds $\underline{V}_t^0$ on V_t
for $k : 0, 1, \dots$ **do**
// Forward Pass : compute a set of trial points $\{x_t^k\}_{t \in [\![0,T]\!]}$
Draw a noise scenario $\{\xi_t^k\}_{t \in [\![1,T]\!]}$
for $t : 0$ **to** $T-1$ **do**
select $x_{t+1}^k \in \arg\min \widehat{\mathcal{T}}_t(\underline{V}_{t+1}^k)(x_t^k, \xi_{t+1}^k)$
end for
// Backward Pass : refine the lower-approximations at the trial points
for $t : T-1$ **to** 0 **do**
for $\xi \in \mathrm{supp}(\boldsymbol{\xi}_{t+1})$ **do**
$\underline{\theta}_t^{\xi,k+1} \leftarrow \hat{\mathcal{T}}_t(\underline{V}_{t+1}^{k+1})(x_t^k, \xi)$
$\underline{\lambda}_t^{\xi,k+1} \in \partial \hat{\mathcal{T}}_t(\underline{V}_{t+1}^{k+1})(x_t^k, \xi)$
end for
$$\underline{\lambda}_t^{k+1} \leftarrow \sum_{\xi \in \mathrm{supp}(\boldsymbol{\xi}_{t+1})} \pi_{t+1}^{\xi} \underline{\lambda}_t^{\xi,k+1}$$
$$\underline{\beta}_t^{k+1} \leftarrow \sum_{\xi \in \mathrm{supp}(\boldsymbol{\xi}_{t+1})} \pi_{t+1}^{\xi} \big(\underline{\theta}_t^{\xi,k+1} - \langle \underline{\lambda}_t^{\xi,k+1}, x_t^k \rangle\big)$$
$\underline{V}_t^{k+1} \leftarrow \max\{\underline{V}_t^k, \langle \underline{\lambda}_t^k, \cdot \rangle + \underline{\beta}_t^{k+1}\}$ // update lower approximation
end for
STOP if some stopping test is satisfied
end for

Contents

Fenchel transform of LBO

Theorem

Assume that the pair $(\mathcal{B}, R)$ satisfy the RCR assumption, R being proper polyhedral, and $\mathcal{B}$ compact (i.e. $\mathcal{G}$ is compact valued with compact domain).

Then $\mathcal{B}(R)$ is a proper function and we have that

$$[\mathcal{B}(R)]^\star = \mathcal{B}^\ddagger(R^\star)$$

where $\mathcal{B}^\ddagger$ is an explicitely given LBO.

Dual LBO

More precisely we have

$$\begin{aligned}
\mathcal{B}^{\ddagger}(Q) : \lambda \mapsto \inf_{\boldsymbol{\mu} \in \mathcal{L}^0(\mathbb{R}^{n_x}), \boldsymbol{\nu} \in \mathcal{L}^0(\mathbb{R}^{n_c})} \quad & \mathbb{E}\Big[-\boldsymbol{\mu}^\top \boldsymbol{h} + Q(\boldsymbol{\nu}) \Big] \\
s.t. \quad & T^\top \mathbb{E}\big[\boldsymbol{\mu}\big] + \lambda = 0 \\
& \mathcal{W}_u^{\dagger}(\boldsymbol{\mu}) = \boldsymbol{C} \\
& \mathcal{W}_y^{\dagger}(\boldsymbol{\mu}) = \boldsymbol{\nu} \\
& \boldsymbol{\mu} \leq 0
\end{aligned}$$

Contents

Recursion over dual value function

Denote $\mathcal{D}_t := V_t^\star$.

Theorem

Then

$$\begin{cases} \mathcal{D}_T &= K^\star \\ \mathcal{D}_t &= \mathcal{B}^\ddagger_{t,L_{t+1}}(\mathcal{D}_{t+1}) \qquad \forall t \in [\![0, T-1]\!] \end{cases}$$

where $\mathcal{B}^\ddagger_{t,L_{t+1}} := \mathcal{B}^\ddagger_t + \mathbb{I}_{\|\lambda_{t+1}\|_\infty \leq L_{t+1}}$.

This is a Bellman recursion on $\mathcal{D}_t$ instead of V_t.
Further, under easy technical assumptions, $\left\{\mathcal{B}^\ddagger_{t,L_{t+1}}\right\}_{t\in[\![0,T]\!]}$ is a compatible sequence of LBOs, where V_t is L_t-Lipschitz.

What's different with Dual SDDP

- It's exactly the abstract SDDP algorithm applied to the dual Bellman Operator...
- except that we need a starting point, and some bounds on the dual variables.
- the main differences with primal SDDP:
  - need to solve the coupled problem $\mathcal{T}_t^{\ddagger}(\mathcal{D}_{t+1}^k)(\lambda_t^k)$ in the forward pass, instead of a "one-realisation" (hat) version
  - also need to solve the coupled problem in the Backward phase, instead of multiple "one-realisation" version that are averaged

What's different with Dual SDDP

- It's exactly the abstract SDDP algorithm applied to the dual Bellman Operator...
- except that we need a starting point, and some bounds on the dual variables.
- the main differences with primal SDDP:
  - need to solve the coupled problem $\mathcal{T}_t^{\ddagger}(\mathcal{D}_{t+1}^k)(\lambda_t^k)$ in the forward pass, instead of a "one-realisation" (hat) version
  - also need to solve the coupled problem in the Backward phase, instead of multiple "one-realisation" version that are averaged

What's different with Dual SDDP

- It's exactly the abstract SDDP algorithm applied to the dual Bellman Operator...
- except that we need a starting point, and some bounds on the dual variables.

- the main differences with primal SDDP:
  - need to solve the coupled problem $\mathcal{T}_t^{\ddagger}(\mathcal{D}_{t+1}^k)(\lambda_t^k)$ in the forward pass, instead of a "one-realisation" (hat) version
  - also need to solve the coupled problem in the Backward phase, instead of multiple "one-realisation" version that are averaged

Data: Initial primal point x_0, Lipschitz bounds $\{L_t\}_{t\in[\![0,T]\!]}$

$\underline{\mathcal{D}}_t^{(0)} \leftarrow -\infty$

for $k : 0,\ 1, \dots$ **do**

// Forward Pass : compute a set of trial points $\{\lambda_t^{(k)}\}_{t\in[\![0,T]\!]}$

Select $\lambda_0^k \in \arg\max_{\|\lambda_0\|_\infty \le L_0} \Big\{ x_0^\top \lambda_0 - \underline{\mathcal{D}}_0^k(\lambda_0) \Big\}$ // Fenchel transform

for $t : 0$ **to** $T-1$ **do**

select $\boldsymbol{\lambda}_{t+1}^k \in \arg\min \mathcal{T}_{t,L_{t+1}}^{\ddagger}(\underline{\mathcal{D}}_{t+1}^k)(\lambda_t^k)$

and draw a realization λ_{t+1}^k of $\boldsymbol{\lambda}_{t+1}^k$

end for

// Backward Pass : refine the lower-approximations at the trial points

$\underline{\mathcal{D}}_T^k \leftarrow K^\star$.

for $t : T-1$ **to** 0 **do**

$\overline{\theta}_t^{k+1} \leftarrow \mathcal{T}_{t,L_{t+1}}^{\ddagger}(\underline{\mathcal{D}}_{t+1}^{k+1})(\lambda_t^k)$ // computing cut coefficients

select $\overline{x}_t^{k+1} \in \partial \mathcal{T}_{t,L_{t+1}}^{\ddagger}(\underline{\mathcal{D}}_{t+1}^{k+1})(\lambda_t^k)$

$\overline{\beta}_t^{k+1} \leftarrow \overline{\theta}_t^{k+1} - \langle \lambda_t^k , \overline{x}_t^{k+1} \rangle$

Define $\mathcal{C}_t^{k+1} : \lambda \mapsto \langle \overline{x}_t^{k+1} , \lambda \rangle + \overline{\beta}_t^{k+1}$

$\underline{\mathcal{D}}_t^{k+1} \leftarrow \max\{\underline{\mathcal{D}}_t^k, \mathcal{C}_t^{k+1}\}$ // update lower approximation

end for

STOP if some stopping test is satisfied

end for

Contents

Converging upper bound and stopping test

We have

$$\underline{V}_t^k \leq V_t$$

and

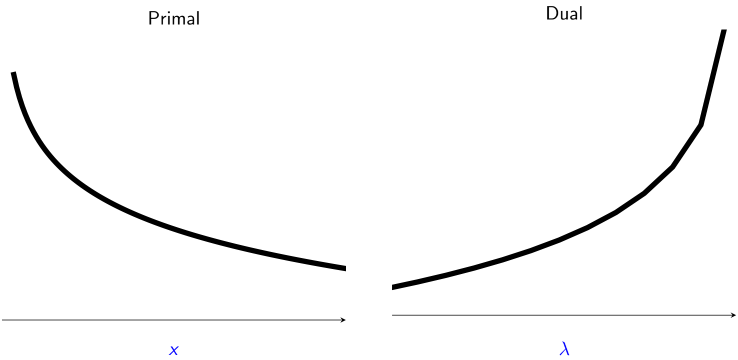
$$\underline{\mathcal{D}}_t^k \leq \mathcal{D}_t \quad \Longrightarrow \quad \underbrace{\left(\underline{\mathcal{D}}_t^k\right)^\star}_{:\approx \overline{V}_t^k} \geq \left(\mathcal{D}_t^\star\right) = V_t^{\star\star} = V_t$$

Finally, we obtain

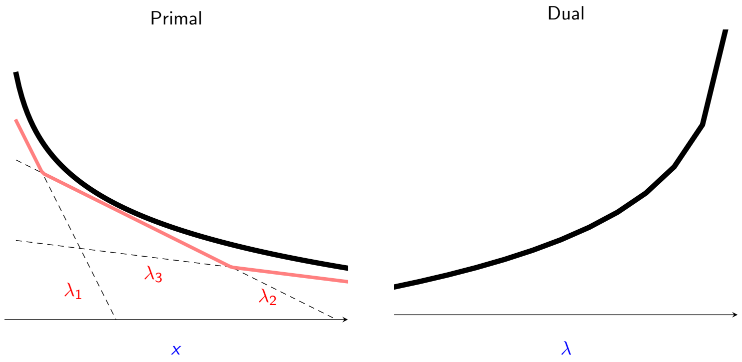
$$\underline{V}_0(x_0) \leq V_0(x_0) \leq \overline{V}_0(x_0).$$

Using the convergence of the abstract SDDP algorithm we show that this bounds are converging, yielding converging deterministic stopping tests.

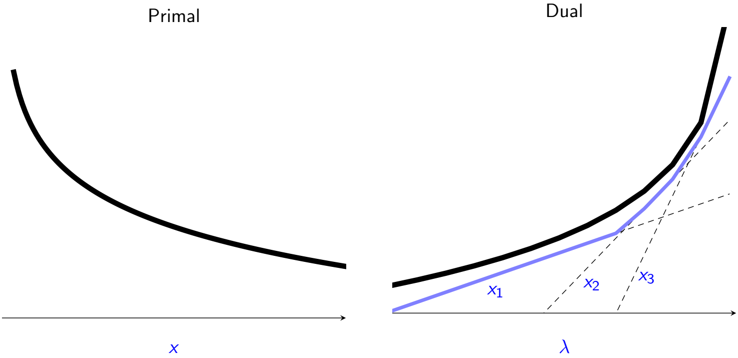
Link between primal and dual approximations



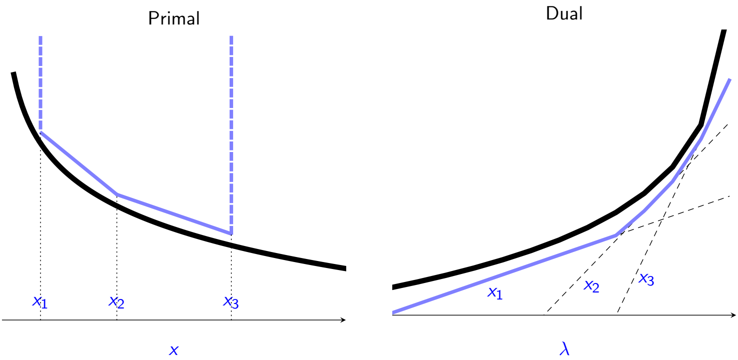
Link between primal and dual approximations



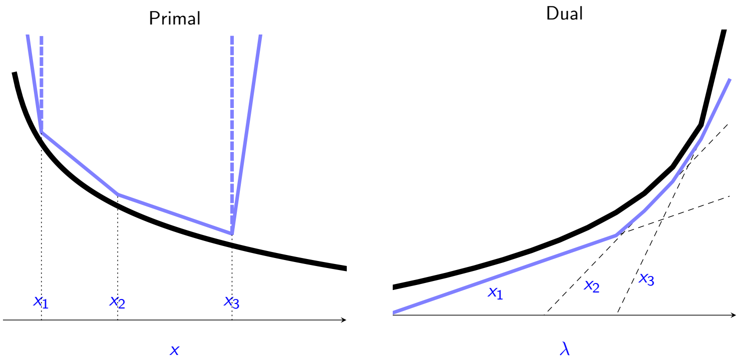
Link between primal and dual approximations



Link between primal and dual approximations



Link between primal and dual approximations



Contents

Inner Approximation

- $\overline{V}_t^k := \left[\underline{\mathcal{D}}_t^k\right]^\star$ which is lower than V_t on X_t
- Or

$$\overline{V}_t^k(x) = \min_{\sigma\in\Delta} \left\{ -\sum_{\kappa=1}^{k} \sigma_\kappa \overline{\beta}_t^\kappa \quad \Big| \quad \sum_{\kappa=1}^{k} \sigma_\kappa \overline{x}_t^\kappa = x \right\}$$

- The inner approximation can be computed by solving

$$\overline{V}_t^{k+1}(x) = \sup_{\lambda,\theta} \quad x^\top \lambda - \theta$$
$$s.t. \quad \theta \geq \left\langle \underline{x}_t^i \, , \lambda \right\rangle + \overline{\beta}_t^\kappa \qquad \forall \kappa \in [\![1, k]\!]$$

Inner Approximation - regularized

- $\overline{V}_t^k := \left[\underline{\mathcal{D}}_t^k\right]^\star \Box (L_t \| \cdot \|_1)$ which is lower than V_t on X_t
- Or

$$\overline{V}_t^k(x) = \min_{y \in \mathbb{R}^{n_x}, \sigma \in \Delta} \left\{ L_t \|x - y\|_1 - \sum_{\kappa=1}^k \sigma_\kappa \overline{\beta}_t^\kappa \quad \middle| \quad \sum_{\kappa=1}^k \sigma_\kappa \overline{x}_t^\kappa = y \right\}$$

- The inner approximation can be computed by solving

$$\begin{aligned} \overline{V}_t^{k+1}(x) = \sup_{\lambda,\theta} \quad & x^\top \lambda - \theta \\ s.t. \quad & \theta \geq \langle \underline{x}_t^i, \lambda \rangle + \overline{\beta}_t^\kappa \qquad \forall \kappa \in [\![1, k]\!] \\ & \|\lambda\|_\infty \leq L_t \end{aligned}$$

A converging strategy - with guaranteed payoff

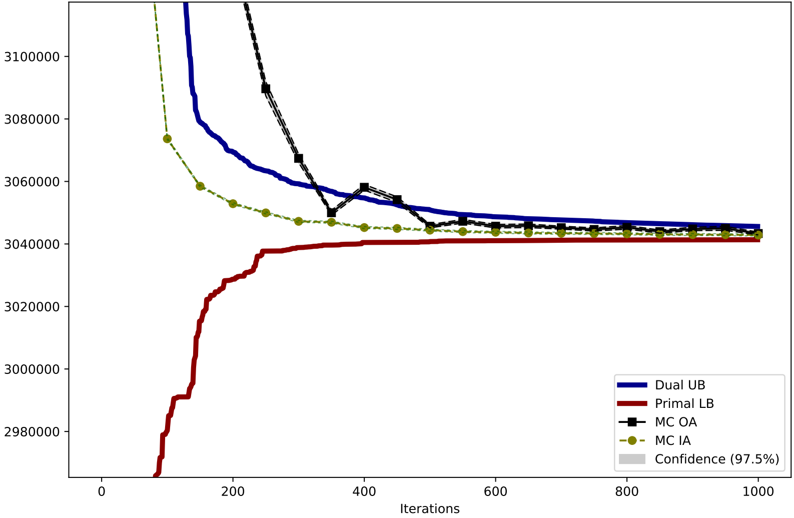
Theorem

Let $C_t^{IA,k}(x)$ be the expected cost of the strategy $\pi^{\overline{V}_t^k}$ when starting from state x at time t.
We have,

$$C_t^{IA,k}(x) \leq \overline{V}_t^k(x) \qquad \lim_k C_t^{IA,k}(x) = V_t(x)$$

Thus, the inner-approximation yields a new converging strategy, and we have an upper-bound on the (expected) value of this strategy.

Numerical results



Stopping test

	Dual stopping test		Statistical stopping test	
ε (%)	n it.	CPU time	n it.	CPU time
2.0	156	183s	250	618s
1.0	236	400s	300	787s
0.5	388	1116s	450	1429s
0.1	$>$ 1000	.	1000	5519s

Table: Comparing dual and statistical stopping criteria for different accuracy levels ε.

Conclusion

- We extend the SDDP algorithm to an abstract framework.
- Leveraging Fenchel conjugate we are able to show a dynamic recursion between dual Bellman value functions.
- We can apply SDDP to this dual recursion.
- This yields a converging exact upper bound on the value of the original problem, hence giving exact and converging stopping tests.
- This also yields a converging strategy with guaranteed payoff.

More information : http://www.optimization-online.org/DB_FILE/2018/04/6575.pdf

Conclusion

- We extend the SDDP algorithm to an abstract framework.
- Leveraging Fenchel conjugate we are able to show a dynamic recursion between dual Bellman value functions.
- We can apply SDDP to this dual recursion.
- This yields a converging exact upper bound on the value of the original problem, hence giving exact and converging stopping tests.
- This also yields a converging strategy with guaranteed payoff.

More information : http://www.optimization-online.org/DB_FILE/2018/04/6575.pdf

Conclusion

- We extend the SDDP algorithm to an abstract framework.
- Leveraging Fenchel conjugate we are able to show a dynamic recursion between dual Bellman value functions.
- We can apply SDDP to this dual recursion.
- This yields a converging exact upper bound on the value of the original problem, hence giving exact and converging stopping tests.
- This also yields a converging strategy with guaranteed payoff.

More information : http://www.optimization-online.org/DB_FILE/2018/04/6575.pdf

Conclusion

- We extend the SDDP algorithm to an abstract framework.
- Leveraging Fenchel conjugate we are able to show a dynamic recursion between dual Bellman value functions.
- We can apply SDDP to this dual recursion.
- This yields a converging exact upper bound on the value of the original problem, hence giving exact and converging stopping tests.
- This also yields a converging strategy with guaranteed payoff.

More information : http://www.optimization-online.org/DB_FILE/2018/04/6575.pdf

Conclusion

- We extend the SDDP algorithm to an abstract framework.
- Leveraging Fenchel conjugate we are able to show a dynamic recursion between dual Bellman value functions.
- We can apply SDDP to this dual recursion.
- This yields a converging exact upper bound on the value of the original problem, hence giving exact and converging stopping tests.
- This also yields a converging strategy with guaranteed payoff.

More information : http://www.optimization-online.org/DB_FILE/2018/04/6575.pdf

Implementation

- We developp an open-source software in the recent, fast and efficient julia language
- You can test Julia through www.juliabox.com
- This software focus on solving stochastic dynamic problem through dynamic programming and SDDP
- You can install the package in Julia simply by typing

```
Pkg.add("StochDynamicProgramming")
```

  in your Julia console
- You can test the dual version through https://github.com/frapac/DualSDDP.jl

Bibliography

- Mario VF Pereira and Leontina MVG Pinto.
  Multi-stage stochastic optimization applied to energy planning.
  Mathematical programming, 52(1-3):359–375, 1991.

- Tito Homem-de Mello, Vitor L De Matos, and Erlon C Finardi.
  Sampling strategies and stopping criteria for stochastic dual dynamic programming: a case study in long-term hydrothermal scheduling.
  Energy Systems, 2(1):1–31, 2011.

- Andrew Philpott, Vitor de Matos, and Erlon Finardi.
  On solving multistage stochastic programs with coherent risk measures.
  Operations Research, 61(4):957–970, 2013.

- Alexander Shapiro.
  Analysis of stochastic dual dynamic programming method.
  European Journal of Operational Research, 209(1):63–72, 2011.

- Pierre Girardeau, Vincent Leclere, and Andrew B Philpott.
  On the convergence of decomposition methods for multistage stochastic convex programs.
  Mathematics of Operations Research, 40(1):130–145, 2014.

Bibliography

Regan Baucke, Anthony Downward, and Golbon Zakeri.
A deterministic algorithm for solving multistage stochastic programming problems.
Optimization Online, 2017.

Vincent Guigues.
Dual dynamic programing with cut selection: Convergence proof and numerical experiments.
European Journal of Operational Research, 258(1):47–57, 2017.

Wim Van Ackooij, Welington de Oliveira, and Yongjia Song.
On regularization with normal solutions in decomposition methods for multistage stochastic programming.
Optimization Online, 2017.


Anymore questions for me?