LinearDecisionRules.jl

Bernardo Freitas Paulo da Costa (UFRJ & FGV)

with Joaquim Garcia (PSR)

JuMP-dev 2024, Montréal

Stochastic Programming

A simple model for stochastic programming:

$$\begin{aligned} \min \quad & \mathbb{E}\left[c^\top x\right] \\ \text{s.t.} \quad & Ax = b, \\ & x \geqslant 0. \end{aligned}$$

where

- x is the decision, subject to (random) constraints;
- c are the (possibly random) costs;

Linear Decision Rules

We write the uncertain parameters as functions of an underlying random vector ξ, and allow for the decision to be taken *after observing the realization of* ξ:

$$
\begin{aligned}
\min \quad & \mathbb{E}\left[c(\xi)^\top x(\xi)\right] & \\
\text{s.t.} \quad & Ax(\xi) = b(\xi) & \forall \xi \in \Xi, \\
& x(\xi) \geqslant 0 & \forall \xi \in \Xi.
\end{aligned}
$$

Linear Decision Rules

We write the uncertain parameters as functions of an underlying random vector ξ, and allow for the decision to be taken *after observing the realization of* ξ:

$$\begin{array}{lll} \min & \mathbb{E}\left[c(\xi)^\top x(\xi)\right] & \\ \text{s.t.} & Ax(\xi) = b(\xi) & \forall \xi \in \Xi, \\ & x(\xi) \geqslant 0 & \forall \xi \in \Xi. \end{array}$$

We assume that the constraint matrix is *deterministic*.

Linear Decision Rules

We write the uncertain parameters as linear functions of an underlying random vector ξ, and allow for the decision to be taken *after observing the realization of ξ*:

$$\begin{aligned} \min \quad & \mathbb{E}\left[\xi^\top C^\top x(\xi)\right] & \\ \text{s.t.} \quad & Ax(\xi) = B\xi & \forall \xi \in \Xi, \\ & x(\xi) \geqslant 0 & \forall \xi \in \Xi. \end{aligned}$$

We assume that the constraint matrix is *deterministic*.

Linear Decision Rules

We write the uncertain parameters as linear functions of an underlying random vector ξ, and allow for the decision to be taken *after observing the realization of ξ*:

$$\begin{aligned} \min \quad & \mathbb{E}\left[\xi^\top C^\top x(\xi)\right] & \\ \text{s.t.} \quad & Ax(\xi) = B\xi & \forall \xi \in \Xi, \\ & x(\xi) \geqslant 0 & \forall \xi \in \Xi. \end{aligned}$$

We assume that the constraint matrix is *deterministic*.
We then posit a linear decision rule for x:

$$x(\xi) = X\xi.$$

Linear Decision Rules

We write the uncertain parameters as linear functions of an underlying random vector ξ, and allow for the decision to be taken *after observing the realization of ξ*:

$$\begin{aligned} \min \quad & \mathbb{E}\left[\xi^\top C^\top X \xi\right] & \\ \text{s.t.} \quad & AX\xi = B\xi & \forall \xi \in \Xi, \\ & X\xi \geqslant 0 & \forall \xi \in \Xi. \end{aligned}$$

We assume that the constraint matrix is *deterministic*.
We then posit a linear decision rule for x:

$$x(\xi) = X\xi.$$

Linear Decision Rules

We write the uncertain parameters as linear functions of an underlying random vector ξ, and allow for the decision to be taken *after observing the realization of* ξ:

$$\begin{aligned} \min \quad & \mathbb{E}\left[\xi^\top C^\top X \xi\right] & \\ \text{s.t.} \quad & AX\xi = B\xi & \forall \xi \in \Xi, \\ & X\xi \geqslant 0 & \forall \xi \in \Xi. \end{aligned}$$

We assume that the constraint matrix is *deterministic*.
We then posit a linear decision rule for x:

$$x(\xi) = X\xi.$$

This reduces the flexibility of the “wait-and-see” decision, but allows for a *more tractable* optimization problem.

Linear Decision Rules — Reformulation

If the uncertainty set Ξ is given as the polytope $\{\, \xi : W\xi \geqslant h \,\}$, we can rewrite the optimization problem as a linear program over the decision rule matrix X and auxiliary variables Λ (for the positivity constraints):

$$
\begin{aligned}
\min_{X,\Lambda} \quad & \mathrm{Tr}\left(\mathbb{E}\left[\xi\xi^\top\right] C^\top X\right) \\
\text{s.t.} \quad & AX = B, \\
& X = \Lambda W,\ \Lambda h \geqslant 0,\ \Lambda \geqslant 0.
\end{aligned}
$$

The `LinearDecisionRules.jl` Package

The package `LinearDecisionRules.jl` provides a JuMP extension for modeling Stochastic Programming problems with linear decision rules.

The `LinearDecisionRules.jl` Package

The package `LinearDecisionRules.jl` provides a JuMP extension for modeling Stochastic Programming problems with linear decision rules.

Usage

1. We introduce `LDRModel` as an extension of `JuMP.Model`;

The `LinearDecisionRules.jl` Package

The package `LinearDecisionRules.jl` provides a JuMP extension for modeling Stochastic Programming problems with linear decision rules.

Usage

1. We introduce `LDRModel` as an extension of `JuMP.Model`;
2. The `@variable` macro is extended to allow for the declaration of *uncertainties* as variables in the model;

The `LinearDecisionRules.jl` Package

The package `LinearDecisionRules.jl` provides a JuMP extension for modeling Stochastic Programming problems with linear decision rules.

Usage

1. We introduce `LDRModel` as an extension of `JuMP.Model`;
2. The `@variable` macro is extended to allow for the declaration of *uncertainties* as variables in the model;
3. Attributes `SolvePrimal()` and `SolveDual()` enable and disable the optimization of primal and dual LDR reformulations.

The `LinearDecisionRules.jl` Package

The package `LinearDecisionRules.jl` provides a JuMP extension for modeling Stochastic Programming problems with linear decision rules.

Usage

1. We introduce `LDRModel` as an extension of `JuMP.Model`;
2. The `@variable` macro is extended to allow for the declaration of *uncertainties* as variables in the model;
3. Attributes `SolvePrimal()` and `SolveDual()` enable and disable the optimization of primal and dual LDR reformulations.
4. We provide `get_decision()` to extract the coefficients of the decision rule matrix X in the original variables and uncertainties. A keyword argument `dual` is used for querying dual decision rule.

A toy (energy!) example

```
using JuMP, LinearDecisionRules
using Ipopt, Distributions

demand = 0.3
initial_volume = 0.5

m = LDRModel()
@variable(m, vi == initial_volume)
@variable(m, 0 <= vf <= 1)
@variable(m, gh >= 0.0)
@variable(m, gt >= 0.0)
@variable(m, 0 <= inflow <= 0.2, Uncertainty,
    distribution=Uniform(0, 0.2))

@constraint(m, balance, vf == vi - gh + inflow)
@constraint(m, gt + gh == demand)

@objective(m, Min, gt^2 + vf^2/2 - vf)
```

A toy example (cont.)

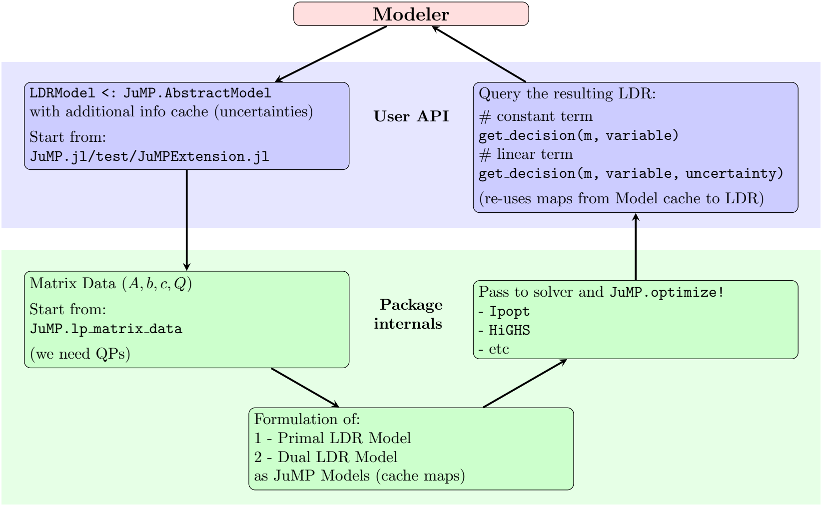
```
# Solve the primal LDR
set_attribute(m, SolvePrimal(), true)
set_attribute(m, SolveDual(), false)
set_optimizer(m, Ipopt.Optimizer)
optimize!(m)

# Get the decision rule
get_decision(m, vf)          # Constant term
get_decision(m, vf, inflow) # Linear coefficient

# Some checks
@test get_decision(m, gh) + get_decision(m, gt) ≈
    demand atol=1e-6
@test get_decision(m, gh, inflow) + get_decision(m
    , gt, inflow) ≈ 0 atol=1e-6

@test get_decision(m, vi) ≈ initial_volume atol=1e
    -6
@test get_decision(m, vi, inflow) ≈ 0 atol=1e-6
```

Package structure



Next steps

Handle *correlated uncertainties*:

- The current model allows for independent uncertainties, and Ξ is the product of their support;
- We could allow for a general form as the product of independent *vector uncertainties*.

Next steps

Handle *correlated uncertainties*:

- The current model allows for independent uncertainties, and Ξ is the product of their support;
- We could allow for a general form as the product of independent *vector uncertainties*.

Multistage decision rules:

- 2-stage optimization: a *here-and-now* decision x_0 which does not depend on uncertainty;

Next steps

Handle *correlated uncertainties*:

- The current model allows for independent uncertainties, and Ξ is the product of their support;
- We could allow for a general form as the product of independent *vector uncertainties*.

Multistage decision rules:

- 2-stage optimization: a *here-and-now* decision x_0 which does not depend on uncertainty;
- In general, decisions x_t can only depend on *observed* uncertainties $\xi_1, \ldots, \xi_t$;
- Will benefit from correlated uncertainties to model more complex processes.

Questions?