The practitioners guide to SDDP

Experiences from `SDDP.jl`

Oscar Dowson

Department of Engineering Science
University of Auckland

June, 2018

Outline

`SDDP.jl`

Scary things with SDDP
- Degeneracy
- Numerical Issues

A crisis of reproducibility

Outline

`SDDP.jl`

Scary things with SDDP

- Degeneracy
- Numerical Issues

A crisis of reproducibility

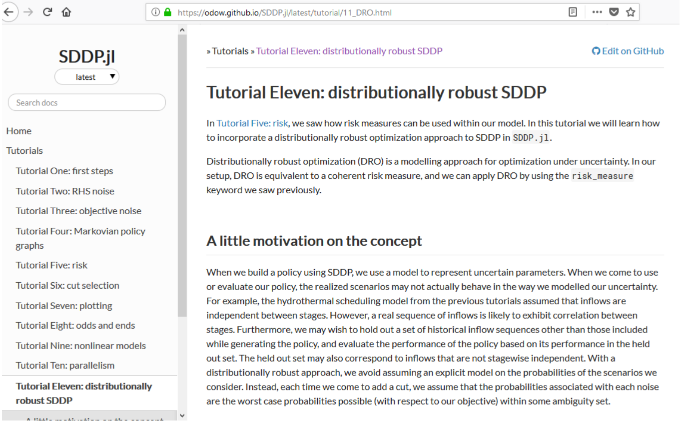
SDDP.jl

What is it?

- A generic SDDP library in the Julia language
- Built upon JuMP $\implies$ nice syntax
- Similar performance to C++ implementation
- Some cool features:
  1. User-extensible risk measures
  2. User-extensible cut selection heuristics
- Google "SDDP.jl github"

SDDP.jl

Success story: the paper of Philpott, de Matos, Kapelevich (2018) used `SDDP.jl` to implement a DRO version of SDDP in < 50 lines of code.



https://odow.github.io/SDDP.jl/latest/tutorial/11_DRO.html

SDDP.jl

latest

Search docs

- Home
- Tutorials
  - Tutorial One: first steps
  - Tutorial Two: RHS noise
  - Tutorial Three: objective noise
  - Tutorial Four: Markovian policy graphs
  - Tutorial Five: risk
  - Tutorial Six: cut selection
  - Tutorial Seven: plotting
  - Tutorial Eight: odds and ends
  - Tutorial Nine: nonlinear models
  - Tutorial Ten: parallelism
  - **Tutorial Eleven: distributionally robust SDDP**

» Tutorials » Tutorial Eleven: distributionally robust SDDP

Edit on GitHub

Tutorial Eleven: distributionally robust SDDP

In Tutorial Five: risk, we saw how risk measures can be used within our model. In this tutorial we will learn how to incorporate a distributionally robust optimization approach to SDDP in `SDDP.jl`.

Distributionally robust optimization (DRO) is a modelling approach for optimization under uncertainty. In our setup, DRO is equivalent to a coherent risk measure, and we can apply DRO by using the `risk_measure` keyword we saw previously.

A little motivation on the concept

When we build a policy using SDDP, we use a model to represent uncertain parameters. When we come to use or evaluate our policy, the realized scenarios may not actually behave in the way we modelled our uncertainty. For example, the hydrothermal scheduling model from the previous tutorials assumed that inflows are independent between stages. However, a real sequence of inflows is likely to exhibit correlation between stages. Furthermore, we may wish to hold out a set of historical inflow sequences other than those included while generating the policy, and evaluate the performance of the policy based on its performance in the held out set. The held out set may also correspond to inflows that are not stagewise independent. With a distributionally robust approach, we avoid assuming an explicit model on the probabilities of the scenarios we consider. Instead, each time we come to add a cut, we assume that the probabilities associated with each noise are the worst case probabilities possible (with respect to our objective) within some ambiguity set.

Outline

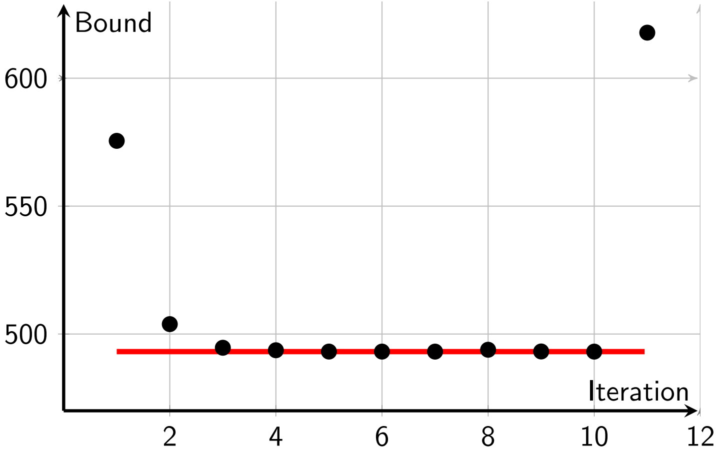
SDDP.jl

Scary things with SDDP

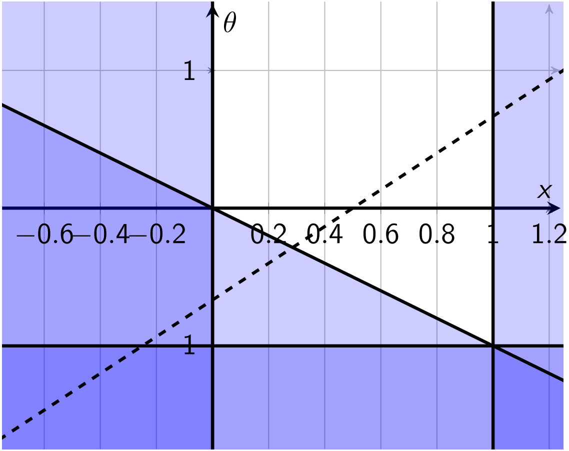
- Degeneracy
- Numerical Issues

A crisis of reproducibility

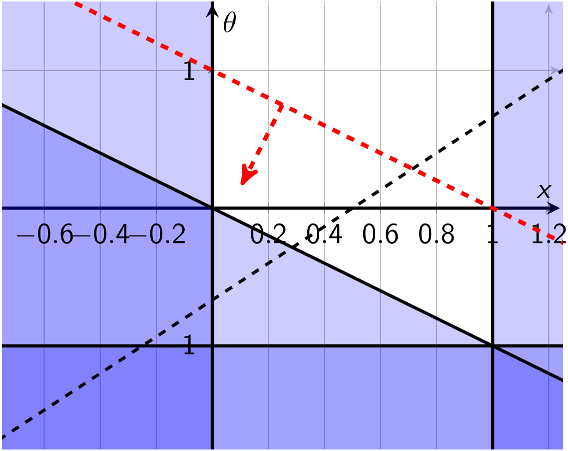
Degeneracy



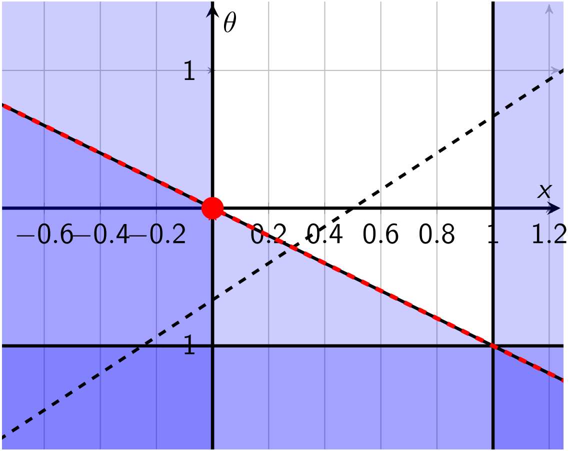
Degeneracy



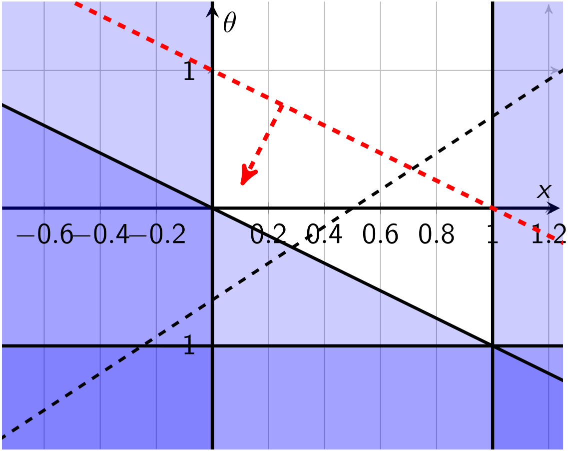
Degeneracy



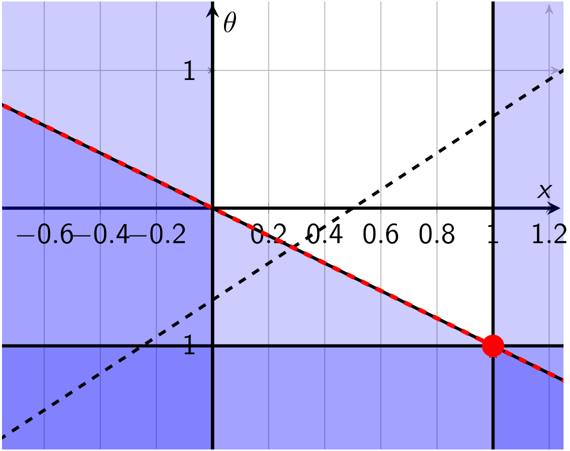
Degeneracy



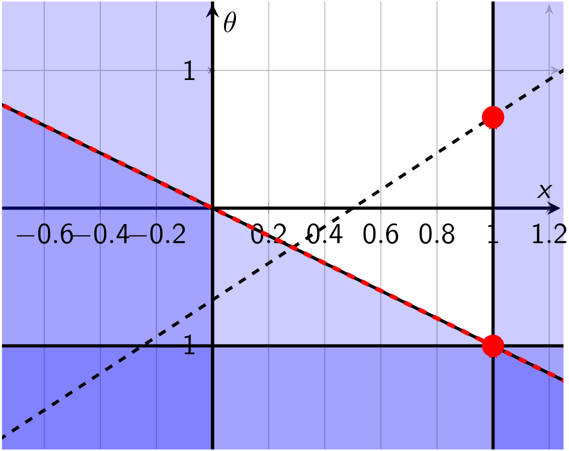
Degeneracy



Degeneracy



Degeneracy



Degeneracy

Implication for SDDP

When we simulate an "optimal" policy, we may obtain a sequence of sub-optimal controls, even in a converged, deterministic model.

Numerical Issues

1. There might be a bug in `SDDP.jl`
2. There might be a bug in your code
3. There might be bugs in JuMP or Julia
4. We have found bugs in Gurobi. Wrong solutions to simple LP's
5. Clp will willingly provide numerically incorrect solutions

Numerical Issues

Implication for SDDP

How do we know a solution is correct? It is hard to validate the solution of a multistage stochastic optimisation problem.

Outline

SDDP.jl

Scary things with SDDP
- Degeneracy
- Numerical Issues

A crisis of reproducibility

A crisis of reproducibility

SDDP is really cool, but you hear the following all the time:

1. I wrote a (private) implementation in XXX language
2. It is hard-coded to solve my favourite (energy-related) problem XXX
3. The standard algorithm is too slow, so I made it XXX times faster by implementing MyNewPublishedTechnique™
4. Everything worked beautifully and I got a nice answer.

A crisis of reproducibility

1. We don't share code
   (so how do I know your implementation is correct?)
2. We can't share models
   (so I can't test my method on your problem)
3. We use the same words with different meanings
   (so I can't easily understand your paper and re-implement)

A crisis of reproducibility

1. We don't share code
   (so how do I know your implementation is correct?)
2. We can't share models
   (so I can't test my method on your problem)
3. We use the same words with different meanings
   (so I can't easily understand your paper and re-implement)

How do we know if MyNewPublishedTechnique™ is an improvement?

A proposal

- Let's pick a subset of problems, large enough to be useful, small enough to standardise
- Agree on a common notation, terminology, and model formulation
- Develop a file-format
- Develop a set of open-source solvers that can read the format
- Develop a set of test problems

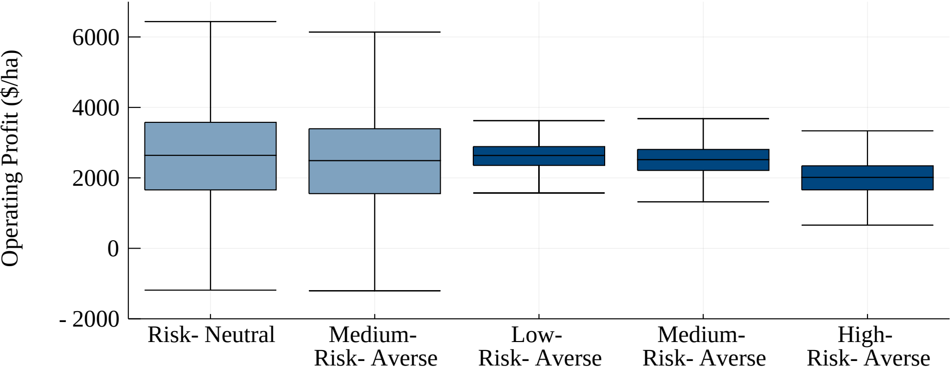
A proposal

- Let's pick a subset of problems, large enough to be useful, small enough to standardise
- Agree on a common notation, terminology, and model formulation
- Develop a file-format
- Develop a set of open-source solvers that can read the format
- Develop a set of test problems
- If MyNewPublishedTechnique$^{\text{TM}}$ can solve the test problems faster it is an improvement.

A proposal

Questions?

Nested Risk Measures



THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Implication for SDDP

Nested risk-measures don't match our intuition, and they don't produce consistent results.