Stochastic Programming for Hydropower Operations
Modeling and Algorithms

Martin Biel
KTH - Royal Institute of Technology
JUNE 28, 2018
Motivation

- Simulation of hydro power operations $\rightarrow$ Decision-support
Motivation

• Simulation of hydro power operations $\rightarrow$ Decision-support
  - Price forecasts
  - Irregular power production: solar and wind
  - Nuclear power phase-out
Motivation

• Simulation of hydro power operations → Decision-support
  ◦ Price forecasts
  ◦ Irregular power production: solar and wind
  ◦ Nuclear power phase-out

• Common: Trade-off between accuracy and computation time
Motivation

• Simulation of hydro power operations → Decision-support
  ◦ Price forecasts
  ◦ Irregular power production: solar and wind
  ◦ Nuclear power phase-out

• Common: Trade-off between accuracy and computation time

• Aim: **Provide reliable decision-support in real time**
Motivation

• Simulation of hydro power operations $\rightarrow$ Decision-support
  ◦ Price forecasts
  ◦ Irregular power production: solar and wind
  ◦ Nuclear power phase-out

• Common: Trade-off between accuracy and computation time

• Aim: Provide reliable decision-support in real time
  ◦ Accurate models: Optimal model reductions
Motivation

- Simulation of hydro power operations $\rightarrow$ Decision-support
  - Price forecasts
  - Irregular power production: solar and wind
  - Nuclear power phase-out

- Common: Trade-off between accuracy and computation time

- Aim: Provide reliable decision-support in real time
  - Accurate models: Optimal model reductions
  - Fast computations: Scalable algorithms on commodity hardware
Motivation

• Simulation of hydro power operations $\rightarrow$ Decision-support
  ◦ Price forecasts
  ◦ Irregular power production: solar and wind
  ◦ Nuclear power phase-out

• Common: Trade-off between accuracy and computation time

• Aim: **Provide reliable decision-support in real time**
  ◦ Accurate models: Optimal model reductions
  ◦ Fast computations: Scalable algorithms on commodity hardware
Motivation

- Simulation of hydro power operations → Decision-support
  - Price forecasts
  - Irregular power production: solar and wind
  - Nuclear power phase-out

- Common: Trade-off between accuracy and computation time

- Aim: **Provide reliable decision-support in real time**
  - Accurate models: Optimal model reductions
  - Fast computations: **Scalable algorithms on commodity hardware**
Motivation

Stochastic programming for hydro power operations

- Optimal orders on the day-ahead market
- Maintenance scheduling
- Long-term investments
- Wind/solar uncertainties
Motivation

Stochastic programming for hydro power operations

- Optimal orders on the day-ahead market
- Maintenance scheduling
- Long-term investments
- Wind/solar uncertainties

Advantages

- Multiple scenarios → More accurate models
- Parallel decomposition → Faster computations
Contribution

Julia modules

- StochasticPrograms.jl
- LShapedSolvers.jl
- HydroModels.jl
Contribution

Julia modules

- StochasticPrograms.jl
- LShapedSolvers.jl
- HydroModels.jl

Software Innovations

- Deferred model creation
- Data injection
Content

- Initial approach
Content

- Initial approach
- StochasticProgramming.jl
Content

- Initial approach
- StochasticProgramming.jl
- LShapedSolvers.jl
Content

• Initial approach
• StochasticProgramming.jl
• LShapedSolvers.jl
• HydroModels.jl
Content

- Initial approach
- StochasticProgramming.jl
- LShapedSolvers.jl
- HydroModels.jl
- Final remarks
Initial Approach

- HydroModel
  - Data
  - JuMP model
Initial Approach

- HydroModel
  - Data
  - JuMP model

- Julia struct for each model: ShortTerm, DayAhead
Initial Approach

- HydroModel
  - Data
  - JuMP model

- Julia struct for each model: ShortTerm, DayAhead

- Parallel decomposition: L-shaped on StructJuMP.jl models
Initial Approach

- HydroModel
  - Data
  - JuMP model

- Julia struct for each model: ShortTerm, DayAhead

- Parallel decomposition: L-shaped on StructJuMP.jl models

- Performance: Solve extended form using MathProgBase solvers
function define_structjump_problem(model::DayAheadModel)
    model.internalmodels[:structured] = StructuredModel(num_scenarios = numscenarios(model))
    params = model.modeldata
    ..
    @variable(internalmodel, xt_i[t = model.hours] >= 0)
    ..
    for s in 1:numscenarios(model)
        block = StructuredModel(parent = internalmodel, id = s)
        ..
        @variable(block, Q[p = model.plants, q = model.segments, t = model.hours],
            lowerbound = 0, upperbound = params.Q[p, q])
        @variable(block, S[p = model.plants, t = model.hours] >= 0)
        ..
        @expression(block, value_of_stored_water,
            params.λ_f*sum(M[p, hours(model.horizon)]*sum(params.μ[i, 1]
                for i = params.Qd[p])
            for p = model.plants))
        # Define objective
        @objective(block, Max, net_profit + value_of_stored_water)
        ..
        @constraint(block, production[s = model.scenarios, t = model.hours],
            H[s, t] == sum(params.μ[p, q]*Q[s, p, q, t]
                for p = model.plants, q = model.segments)
            )
    ..
end
function define_dep_problem(model::DayAheadModel)
    model.internalmodels[:dep] = Model()
    params = model.modeldata
    ...
    @variable(internalmodel, xt_i[t = model.hours] >= 0)
    ...
    @variable(block, Q[s = model.scenarios, p = _model.plants, t = model.hours],
              lowerbound = 0, upperbound = params.Q[(p,q)])
    @variable(block, S[s = model.scenarios, p = model.plants, t = model.hours] >= 0)
    ...
    @expression(block, value_of_stored_water,
                sum(scenarios[s].π*params.λ_f*sum(M[s,p]*sum(params.μ[i,1]
                                                 for i = params.Qd[p])
                                           for p = model.plants)
                  for s = model.scenarios))

    # Define objective
    @objective(block, Max, net_profit + value_of_stored_water)
    ...
    @constraint(block, production[s = model.scenarios, t = model.hours],
                H[s,t] == sum(params.μ[p,q]*Q[s,p,q,t]
                              for p = model.plants, q = model.segments)
            )
    ...
end
Initial Approach - Issues

- A lot of code repetition
Initial Approach - Issues

- A lot of code repetition
- No clearcut way to calculate stochastic measures: EVPI, VSS
Initial Approach - Issues

- A lot of code repetition
- No clearcut way to calculate stochastic measures: EVPI, VSS
- The model creation is somewhat inflexible
Initial Approach - Issues

- A lot of code repetition
- No clearcut way to calculate stochastic measures: EVPI, VSS
- The model creation is somewhat inflexible
- Parallel L-shaped using the Distributed module in Julia...
Initial Approach - Issues

- A lot of code repetition
- No clearcut way to calculate stochastic measures: EVPI, VSS
- The model creation is somewhat inflexible
- Parallel L-shaped using the Distributed module in Julia...
- ...but StructJuMP relies on MPI
Initial Approach - Issues

• A lot of code repetition
• No clearcut way to calculate stochastic measures: EVPI, VSS
• The model creation is somewhat inflexible
• Parallel L-shaped using the Distributed module in Julia…
• …but StructJuMP relies on MPI
• Creating a new hydromodel involves reimplementing a new type
New Approach

- StochasticPrograms.jl

- HydroModels.jl
New Approach

- StochasticPrograms.jl
  - Flexible model creation

- HydroModels.jl
New Approach

• StochasticPrograms.jl
  ◦ Flexible model creation
  ◦ Parallel capabilities based on the Distributed module

• HydroModels.jl
New Approach

- **StochasticPrograms.jl**
  - Flexible model creation
  - Parallel capabilities based on the Distributed module
  - Stochastic programming constructs

- **HydroModels.jl**
New Approach

• StochasticPrograms.jl
  ◦ Flexible model creation
  ◦ Parallel capabilities based on the Distributed module
  ◦ Stochastic programming constructs

• HydroModels.jl
  ◦ Model creation focused on data and optimization formulation
New Approach

- **StochasticPrograms.jl**
  - Flexible model creation
  - Parallel capabilities based on the Distributed module
  - Stochastic programming constructs

- **HydroModels.jl**
  - Model creation focused on data and optimization formulation
  - Efficient model reinitialization
New Approach

- **StochasticPrograms.jl**
  - Flexible model creation
  - Parallel capabilities based on the Distributed module
  - Stochastic programming constructs

- **HydroModels.jl**
  - Model creation focused on data and optimization formulation
  - Efficient model reinitialization
  - Predefined models
    - Short-term production planning
    - Optimal orders on the day-ahead market
minimize \quad 100x_1 + 150x_2 + \mathbb{E}_\omega[Q(x_1, x_2, \xi)]

s.t. \quad x_1 + x_2 \leq 120
\quad x_1 \geq 40
\quad x_2 \geq 20

where

\begin{align*}
Q(x_1, x_2, \xi) &= \min_{y_1, y_2 \in \mathbb{R}} q_1(\xi)y_1 + q_2(\xi)y_2 \\
\text{s.t.} \quad 6y_1 + 10y_2 &\leq 60x_1 \\
8y_1 + 5y_2 &\leq 80x_2 \\
0 &\leq y_1 \leq d_1(\xi) \\
0 &\leq y_2 \leq d_2(\xi)
\end{align*}
StochasticPrograms.jl - Simple Example

```julia
sp = StochasticProgram(solver=ClpSolver())

@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁+x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    s = scenario
    @variable(model, 0 <= y₁ <= s.d[1])
    @variable(model, 0 <= y₂ <= s.d[2])
    @objective(model, Min, s.q[1]*y₁ + s.q[2]*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end
```
StochasticPrograms.jl - Simple Example

sp = StochasticProgram(solver=ClpSolver())

@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁+x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    s = scenario
    @variable(model, 0 <= y₁ <= s.d[1])
    @variable(model, 0 <= y₂ <= s.d[2])
    @objective(model, Min, s.q[1]*y₁ + s.q[2]*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end

Creates a generator function for the first stage model
StochasticPrograms.jl - Simple Example

sp = StochasticProgram(solver=ClpSolver())

@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁+x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    s = scenario
    @variable(model, 0 <= y₁ <= s.d[1])
    @variable(model, 0 <= y₂ <= s.d[2])
    @objective(model, Min, s.q[1]*y₁ + s.q[2]*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end

Creates a generator function for the second stage model
StochasticPrograms.jl - Simple Example

sp = StochasticProgram(solver=ClpSolver())

@first_stage sp = begin
  @variable(model, x₁ >= 40)
  @variable(model, x₂ >= 20)
  @objective(model, Min, 100*x₁ + 150*x₂)
  @constraint(model, x₁+x₂ <= 120)
end

@second_stage sp = begin
  @decision x₁ x₂
  s = scenario
  @variable(model, 0 <= y₁ <= s.d[1])
  @variable(model, 0 <= y₂ <= s.d[2])
  @objective(model, Min, s.q[1]*y₁ + s.q[2]*y₂)
  @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
  @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end

Explicitly denote that some variables originate from the first stage
sp = StochasticProgram(solver=ClpSolver())

@first_stage sp = begin
    @variable(model, x₁ >= 40)
    @variable(model, x₂ >= 20)
    @objective(model, Min, 100*x₁ + 150*x₂)
    @constraint(model, x₁+x₂ <= 120)
end

@second_stage sp = begin
    @decision x₁ x₂
    s = scenario
    @variable(model, 0 <= y₁ <= s.d[1])
    @variable(model, 0 <= y₂ <= s.d[2])
    @objective(model, Min, s.q[1]*y₁ + s.q[2]*y₂)
    @constraint(model, 6*y₁ + 10*y₂ <= 60*x₁)
    @constraint(model, 8*y₁ + 5*y₂ <= 80*x₂)
end

Injection point for scenario data
StochasticPrograms.jl - Simple Example

```julia
struct SimpleScenario <: AbstractScenarioData
    p::Float64
    d::Vector{Float64}
    q::Vector{Float64}
end

StochasticPrograms.probability(s::SimpleScenario) = s.p
```
StochasticPrograms.jl - Simple Example

```julia
struct SimpleScenario <: AbstractScenarioData
    p::Float64
    d::Vector{Float64}
    q::Vector{Float64}
end

StochasticPrograms.probability(s::SimpleScenario) = s.p

Add two scenarios to the stochastic program

s1 = SimpleScenario(0.4,[500.0,100],[-24.0,-28])
s2 = SimpleScenario(0.6,[300.0,300],[-28.0,-32])
append!(sp,[s1,s2])
```
StochasticPrograms.jl - Simple Example

print(sp)
StochasticPrograms.jl - Simple Example

print(sp)

First-stage
============
Min 100 \times_1 + 150 \times_2
Subject to
\times_1 + \times_2 \leq 120
\times_1 \geq 40
\times_2 \geq 20

Second-stage
=============
Subproblem 1:
Min -24y_1 - 28y_2
Subject to
6y_1 + 10y_2 - 60\times_1 \leq 0
8y_1 + 5y_2 - 80\times_2 \leq 0
0 \leq y_1 \leq 500
0 \leq y_2 \leq 100

Subproblem 2:
Min -28y_1 - 32y_2
Subject to
6y_1 + 10y_2 - 60\times_1 \leq 0
8y_1 + 5y_2 - 80\times_2 \leq 0
0 \leq y_1 \leq 300
0 \leq y_2 \leq 300
Implementation Details

Deferred model creation

• JuMP models are not created instantly
• Model definitions are stored in generating lambda functions
• These model recipes are then used as building blocks

Data injection
• The generating functions contain certain placeholders keywords
• Upon model creation, the keywords contain the required data fields

Implications
• Flexible model creation and reformulation
• Efficient parallel implementation
• Versatility
Implementation Details

Deferred model creation

- JuMP models are not created instantly

Data injection

- Generating functions contain certain placeholders keywords
- Upon model creation, the keywords contain the required data fields

Implications

- Flexible model creation and reformulation
- Efficient parallel implementation
- Versatility
Implementation Details

Deferred model creation

- JuMP models are not created instantly
- Model definitions are stored in generating lambda functions

Implications

- Flexible model creation and reformulation
- Efficient parallel implementation
- Versatility
Implementation Details

Deferred model creation

- JuMP models are not created instantly
- Model definitions are stored in generating lambda functions
- These model recipes are then used as building blocks
Implementation Details

Deferred model creation

- JuMP models are not created instantly
- Model definitions are stored in generating lambda functions
- These model recipes are then used as building blocks

Data injection
Implementation Details

Deferred model creation

- JuMP models are not created instantly
- Model definitions are stored in generating lambda functions
- These model recipes are then used as building blocks

Data injection

- The generating functions contain certain placeholders keywords
Implementation Details

Deferred model creation

- JuMP models are not created instantly
- Model definitions are stored in generating lambda functions
- These model recipes are then used as building blocks

Data injection

- The generating functions contain certain placeholders keywords
- Upon model creation, the keywords contain the required data fields
Implementation Details

Deferred model creation

- JuMP models are not created instantly
- Model definitions are stored in generating lambda functions
- These model recipes are then used as building blocks

Data injection

- The generating functions contain certain placeholders keywords
- Upon model creation, the keywords contain the required data fields

Implications
Implementation Details

Deferred model creation

- JuMP models are not created instantly
- Model definitions are stored in generating lambda functions
- These model recipes are then used as building blocks

Data injection

- The generating functions contain certain placeholders keywords
- Upon model creation, the keywords contain the required data fields

Implications

- Flexible model creation and reformulation
Implementation Details

Deferred model creation

- JuMP models are not created instantly
- Model definitions are stored in generating lambda functions
- These model recipes are then used as building blocks

Data injection

- The generating functions contain certain placeholders keywords
- Upon model creation, the keywords contain the required data fields

Implications

- Flexible model creation and reformulation
- Efficient parallel implementation
Implementation Details

Deferred model creation

- JuMP models are not created instantly
- Model definitions are stored in generating lambda functions
- These model recipes are then used as building blocks

Data injection

- The generating functions contain certain placeholders keywords
- Upon model creation, the keywords contain the required data fields

Implications

- Flexible model creation and reformulation
- Efficient parallel implementation
- Versatility
\[
\text{minimize } \quad c^T x + \mathbb{E}_\omega [Q(x, \xi(\omega))] \\
\text{s.t. } \quad Ax = b
\]
minimize \( x \in \mathbb{R}^n \)
\[ c^T x + \mathbb{E}_\omega [Q(x, \xi(\omega))] \]

s.t. \( Ax = b \)

dep = DEP(sp)

Minimization problem with:
* 5 linear constraints
* 6 variables
Solver is ClpMathProg
minimize \quad c^T x + \mathbb{E}\omega[Q(x, \xi(\omega))] \\
\text{s.t.} \quad Ax = b \\
\text{dep} = \text{DEP}(sp) \\
\text{Minimization problem with:} \\
\ast \ 5 \text{ linear constraints} \\
\ast \ 6 \text{ variables} \\
\text{Solver is ClpMathProg} \\
\bullet \text{ First stage generator}
**minimize** \( c^T x + \mathbb{E}_\omega [Q(x, \xi(\omega))] \)

s.t. \( Ax = b \)

\[ \text{dep} = \text{DEP}(sp) \]

Minimization problem with:
* 5 linear constraints
* 6 variables
Solver is ClpMathProg

- First stage generator
- Second stage generator on all available scenarios
minimize \( c^T x + \mathbb{E}_\omega [Q(x, \xi(\omega))] \)

s.t. \( Ax = b \)

dep = DEP(sp)

Minimization problem with:
* 5 linear constraints
* 6 variables
Solver is ClpMathProg

- First stage generator
- Second stage generator on all available scenarios
- Connections possible due to the @decision annotation
minimize \[ c^T x + \mathbb{E}_\omega [Q(x, \xi(\omega))] \]

s.t. \[ Ax = b \]

dep = DEP(sp)

Minimization problem with:
* 5 linear constraints
* 6 variables
Solver is ClpMathProg

- First stage generator
- Second stage generator on all available scenarios
- Connections possible due to the `@decision` annotation
- DEP model is cached internally until new scenarios are added
print(dep)
print(dep)

Min 100 \( x_1 \) + 150 \( x_2 \) - 9.6 \( y_{1\_1} \) - 11.2 \( y_{2\_1} \) - 16.8 \( y_{1\_2} \) - 19.2 \( y_{2\_2} \)

Subject to
\[ x_1 + x_2 \leq 120 \]
\[ 6 \ y_{1\_1} + 10 \ y_{2\_1} - 60 \ x_1 \leq 0 \]
\[ 8 \ y_{1\_1} + 5 \ y_{2\_1} - 80 \ x_2 \leq 0 \]
\[ 6 \ y_{1\_2} + 10 \ y_{2\_2} - 60 \ x_1 \leq 0 \]
\[ 8 \ y_{1\_2} + 5 \ y_{2\_2} - 80 \ x_2 \leq 0 \]
\[ x_1 \geq 40 \]
\[ x_2 \geq 20 \]
\[ 0 \leq y_{1\_1} \leq 500 \]
\[ 0 \leq y_{2\_1} \leq 100 \]
\[ 0 \leq y_{1\_2} \leq 300 \]
\[ 0 \leq y_{2\_2} \leq 300 \]
StochasticPrograms.jl - Solving Models

• Extended form
  \[ \text{solve}(sp, \text{solver} = \text{ClpSolver()}) \]
  Optimal
  \[ \text{getobjectivevalue}(sp) \]
  \(-855.83\)

• L-shaped
  \[ \text{solve}(sp, \text{solver} = \text{LShapedSolver}(:ls, \text{ClpSolver()}) \]
  L-Shaped Gap Time : 0:00:01 (4 iterations)
  Objective: \(-855.8333333333358\)
  Gap: \(2.1229209144670507 \times 10^{-15}\)
  Number of cuts: 5
  Optimal

• Convenience function (Value of the recourse problem)
  \[ \text{VRP}(sp, \text{solver} = \text{ClpSolver()}) \]
  \(-855.83\)
• Extended form

```
solve(sp, solver=ClpSolver())
:Optimal

getobjectivevalue(sp)
-855.83
```
• **Extended form**

```julia
solve(sp,solver=ClpSolver())
:Optimal

gteobjectivevalue(sp)
  -855.83
```

• **L-shaped**

```julia
solve(sp,solver=LShapedSolver(:ls,ClpSolver()))
```

L-Shaped Gap  Time: 0:00:01 (4 iterations)
  Objective: -855.8333333333358
  Gap: 2.1229209144670507e-15
  Number of cuts: 5
:Optimal
- Extended form

```python
solve(sp, solver=ClpSolver())
: Optimal

getobjectivevalue(sp)
-855.83
```

- L-shaped

```python
solve(sp, solver=LShapedSolver(:ls, ClpSolver()))
```

```
L-Shaped Gap  Time: 0:00:01 (4 iterations)
Objective: -855.83333333333358
Gap: 2.1229209144670507e-15
Number of cuts: 5
: Optimal
```

- Convenience function (Value of the recourse problem)

```python
VRP(sp, solver=ClpSolver())
-855.83
```
StochasticPrograms.jl - Wait-And-See Models

\[
\begin{align*}
\text{minimize} & \quad c^T x + Q(x, \tilde{\xi}) \\
\text{s.t.} & \quad Ax = b \\
& \quad x \geq 0
\end{align*}
\]

for given $\tilde{\xi}$
minimize \( x \in \mathbb{R}^n \) 
\[ c^T x + Q(x, \tilde{\xi}) \]

s.t. \( Ax = b \)
\[ x \geq 0 \]

for given \( \tilde{\xi} \)

ws = WS(sp,s1)

Minimization problem with:
* 3 linear constraints
* 4 variables

Solver is ClpMathProg
minimize \( c^T x + Q(x, \xi) \)

\[ x \in \mathbb{R}^n \]

s.t. \( Ax = b \)

\[ x \geq 0 \]

for given \( \xi \)

ws = WS(sp,s1)

Minimization problem with:
* 3 linear constraints
* 4 variables

Solver is ClpMathProg

- First stage generator
minimize \( c^T x + Q(x, \tilde{\xi}) \)

s.t.  \( Ax = b \)

\( x \geq 0 \)

for given \( \tilde{\xi} \)

ws = WS(sp,sp)

Minimization problem with:

* 3 linear constraints
* 4 variables

Solver is ClpMathProg

- First stage generator
- Second stage generator on the given scenario
StochasticPrograms.jl - Wait-And-See Models

print(ws)
print(ws)

Min 100 \( x_1 \) + 150 \( x_2 \) - 24 \( y_1 \) - 28 \( y_2 \)
Subject to
\( x_1 + x_2 \leq 120 \)
\( 6 \ y_1 + 10 \ y_2 - 60 \ x_1 \leq 0 \)
\( 8 \ y_1 + 5 \ y_2 - 80 \ x_2 \leq 0 \)
\( x_1 \geq 40 \)
\( x_2 \geq 20 \)
\( 0 \leq y_1 \leq 500 \)
\( 0 \leq y_2 \leq 100 \)
minimize \( c^T x + Q(x, \bar{\xi}) \)

s.t. \( Ax = b \)

\( x \geq 0 \)

where

\( \bar{\xi} = \mathbb{E}_\omega[\xi(\omega)] \)
minimize \( x \in \mathbb{R}^n \) \( c^T x + Q(x, \bar{\xi}) \)

s.t. \( Ax = b \)
\( x \geq 0 \)

where

\[ \bar{\xi} = \mathbb{E}_\omega[\xi(\omega)] \]

Must be possible to take expectation over scenarios

```python
function expected(scenarios::Vector{SimpleScenario})
    return SimpleScenario(sum([s.p for s in scenarios]),
                           sum([s.p*s.d for s in scenarios]),
                           sum([s.p*s.q for s in scenarios]))
end
```
evp = EVP(sp)
Minimization problem with:
* 3 linear constraints
* 4 variables
Solver is ClpMathProg

print(evp)
evp = EVP(sp)
Minimization problem with:
* 3 linear constraints
* 4 variables
Solver is ClpMathProg

print(evp)

Min $100 \ x_1 + 150 \ x_2 - 26.4 \ y_1 - 30.4 \ y_2$
Subject to
\begin{align*}
x_1 + x_2 & \leq 120 \\
6 \ y_1 + 10 \ y_2 - 60 \ x_1 & \leq 0 \\
8 \ y_1 + 5 \ y_2 - 80 \ x_2 & \leq 0 \\
x_1 & \geq 40 \\
x_2 & \geq 20 \\
0 & \leq y_1 \leq 380 \\
0 & \leq y_2 \leq 220
\end{align*}
\[ c^T \hat{x} + \mathbb{E}_\omega [Q(\hat{x}, \xi(\omega))] \]
\[
c^T \hat{x} + \mathbb{E}_\omega [Q(\hat{x}, \xi(\omega))] \]

\(\hat{x} = [50, 50];\)

eval\_decision(sp, \hat{x}, solver=ClpSolver())

356.0

- Create first stage variables using generator
\[ c^T \hat{x} + \mathbb{E}_\omega [Q(\hat{x}, \xi(\omega))] \]

\[ \hat{x} = [50, 50]; \]

eval\_decision(sp, \hat{x}, solver=ClpSolver())

356.0

- Create first stage variables using generator
- Fixate them to the given values
\[ c^T \hat{x} + \mathbb{E}_\omega [Q(\hat{x}, \xi(\omega))] \]

\[ \hat{x} = [50, 50]; \]

eval\_decision(sp, \hat{x}, solver=ClpSolver())

356.0

- Create first stage variables using generator
- Fixate them to the given values
- Generate the second stage problems
\[ c^T \hat{x} + \mathbb{E}_\omega [Q(\hat{x}, \xi(\omega))] \]

\[ \hat{x} = [50, 50]; \]

eval_decision(sp, \hat{x}, solver=ClpSolver())

356.0

- Create first stage variables using generator
- Fixate them to the given values
- Generate the second stage problems
- Again, linking handled through \texttt{@decision}
\[ c^T \hat{x} + \mathbb{E}_\omega [Q(\hat{x}, \xi(\omega))] \]

\( \hat{x} = [50, 50]; \)

eval\_decision(sp, \hat{x}, solver=ClpSolver())

356.0

- Create first stage variables using generator
- Fixate them to the given values
- Generate the second stage problems
- Again, linking handled through @decision
- Solve resulting JuMP model
StochasticPrograms.jl - Stochastic Measures

- Expected value of using the expected solution (EEV)
  \[ \text{EEV}(sp, \text{solver} = \text{ClpSolver()}) = -568.92 \]

- Expected wait-and-see solution (EWS)
  \[ \text{EWS}(sp, \text{solver} = \text{ClpSolver()}) = -1518.75 \]

- Expected value of perfect information (EVPI = VRP - EWS)
  \[ \text{EVPI}(sp, \text{solver} = \text{ClpSolver()}) = 662.92 \]

- Value of the stochastic solution (VSS = EEV - VRP)
  \[ \text{VSS}(sp, \text{solver} = \text{ClpSolver()}) = 286.92 \]
• Expected value of using the expected solution (EEV)
  \[ \text{EEV}(sp, \text{solver} = \text{ClpSolver()}) \]
  \[-568.92\]

• Expected wait-and-see solution (EWS)
  \[ \text{EWS}(sp, \text{solver} = \text{ClpSolver()}) \]
  \[-1518.75\]
• Expected value of using the expected solution (EEV)
  \[ \text{EEV}(\text{sp}, \text{solver}=\text{ClpSolver}()) = -568.92 \]

• Expected wait-and-see solution (EWS)
  \[ \text{EWS}(\text{sp}, \text{solver}=\text{ClpSolver}()) = -1518.75 \]

• Expected value of perfect information (EVPI = VRP - EWS)
  \[ \text{EVPI}(\text{sp}, \text{solver}=\text{ClpSolver}()) = 662.92 \]
Stochastic Programs.jl - Stochastic Measures

- Expected value of using the expected solution (EEV)
  \[ \text{EEV}(\text{sp}, \text{solver} = \text{ClpSolver()}) \]
  \[-568.92\]

- Expected wait-and-see solution (EWS)
  \[ \text{EWS}(\text{sp}, \text{solver} = \text{ClpSolver()}) \]
  \[-1518.75\]

- Expected value of perfect information (EVPI = VRP - EWS)
  \[ \text{EVPI}(\text{sp}, \text{solver} = \text{ClpSolver()}) \]
  \[662.92\]

- Value of the stochastic solution (VSS = EEV - VRP)
  \[ \text{VSS}(\text{sp}, \text{solver} = \text{ClpSolver()}) \]
  \[286.92\]
Expected value of using the expected solution (EEV)
\[ \text{EEV}(sp, \text{solver}=\text{ClpSolver}()) \]
\[ -568.92 \]

Expected wait-and-see solution (EWS)
\[ \text{EWS}(sp, \text{solver}=\text{ClpSolver}()) \]
\[ -1518.75 \]

Expected value of perfect information (EVPI = VRP - EWS)
\[ \text{EVPI}(sp, \text{solver}=\text{ClpSolver}()) \]
\[ 662.92 \]

Value of the stochastic solution (VSS = EEV - VRP)
\[ \text{VSS}(sp, \text{solver}=\text{ClpSolver}()) \]
\[ 286.92 \]

Many of the required calculations are embarrassingly parallel
L-shaped algorithm variants

- L-shaped [Van Slyke, Wets]
- Multicut L-shaped [Birge, Louveaux]
- Regularized decomposition [Ruszczyński]
- Trust-region L-shaped [Linderoth, Wright]
- Level-set [Fábián, Szőke]
LShapedSolvers.jl

- **L-shaped variants**
  1. L-shaped with multiple cuts (default): `LShapedSolver(:ls)`
  2. L-shaped with regularized decomposition: `LShapedSolver(:rd)`
  3. L-shaped with trust region: `LShapedSolver(:tr)`
  4. L-shaped with level sets: `LShapedSolver(:lv)`

- **Distributed L-shaped variants**
  1. Distributed L-shaped with multiple cuts: `LShapedSolver(:dls)`
  2. Distributed regularized L-shaped: `LShapedSolver(:drd)`
  3. Distributed L-shaped with trust region: `LShapedSolver(:dtr)`
  4. Distributed L-shaped with level sets: `LShapedSolver(:dlv)`

- **Trait based implementation.** Every solver is a combination of a:
  - Regularization trait
  - Parallelization trait

- **Subproblems are solved using MathProgBase solvers**
LShapedSolvers.jl

- **L-shaped variants**
  1. L-shaped with multiple cuts (default): LShapedSolver(:ls)
  2. L-shaped with regularized decomposition: LShapedSolver(:rd)
  3. L-shaped with trust region: LShapedSolver(:tr)
  4. L-shaped with level sets: LShapedSolver(:lv)

- **Distributed L-shaped variants**
  1. Distributed L-shaped with multiple cuts: LShapedSolver(:dls)
  2. Distributed regularized L-shaped: LShapedSolver(:drd)
  3. Distributed L-shaped with trust region: LShapedSolver(:dtr)
  4. Distributed L-shaped with level sets: LShapedSolver(:dlv)
• **L-shaped variants**
  1. L-shaped with multiple cuts (default): `LShapedSolver(:ls)`
  2. L-shaped with regularized decomposition: `LShapedSolver(:rd)`
  3. L-shaped with trust region: `LShapedSolver(:tr)`
  4. L-shaped with level sets: `LShapedSolver(:lv)`

• **Distributed L-shaped variants**
  1. Distributed L-shaped with multiple cuts: `LShapedSolver(:dls)`
  2. Distributed regularized L-shaped: `LShapedSolver(:drd)`
  3. Distributed L-shaped with trust region: `LShapedSolver(:dtr)`
  4. Distributed L-shaped with level sets: `LShapedSolver(:dlv)`

• **Trait based implementation.** Every solver is a combination of a:
  ∘ Regularization trait
  ∘ Parallelization trait
**LShapedSolvers.jl**

- **L-shaped variants**
  1. L-shaped with multiple cuts (default): `LShapedSolver(:ls)`
  2. L-shaped with regularized decomposition: `LShapedSolver(:rd)`
  3. L-shaped with trust region: `LShapedSolver(:tr)`
  4. L-shaped with level sets: `LShapedSolver(:lv)`

- **Distributed L-shaped variants**
  1. Distributed L-shaped with multiple cuts: `LShapedSolver(:dls)`
  2. Distributed regularized L-shaped: `LShapedSolver(:drd)`
  3. Distributed L-shaped with trust region: `LShapedSolver(:dtr)`
  4. Distributed L-shaped with level sets: `LShapedSolver(:dlv)`

- **Trait based implementation. Every solver is a combination of a:**
  - Regularization trait
  - Parallelization trait

- **Subproblems are solved using MathProgBase solvers**
HydroModels.jl

- Also based on deferred model creation and data injection
HydroModels.jl

- Also based on deferred model creation and data injection
- The user creates a model recipe using the @hydromodel macro
HydroModels.jl

- Also based on deferred model creation and data injection
- The user creates a model recipe using the `@hydromodel` macro

Creating a Planning Problem

- Define model indices
- Define model data
- Define `modelindices(::AbstractHydroModelData, ::Horizon, args...)`
- Define optimization problem
HydroModels.jl

• Also based on deferred model creation and data injection
• The user creates a model recipe using the @hydromodel macro

Creating a Planning Problem

• Define model indices
• Define model data
• Define `modelindices(::AbstractHydroModelData, ::Horizon, args...)`
• Define optimization problem

Data injection keywords

• `horizon`: the time horizon if the model
• `indices`: structure with model indices
• `data`: structure with model data
HydroModels.jl - Simple Example

```julia
struct SimpleShortTermIndices <: AbstractModelIndices
    hours::Vector{Int}
    plants::Vector{Symbol}
end

struct SimpleShortTermData <: AbstractModelData
    hydrodata::HydroPlantCollection{Float64,2}
    D::Vector{Float64}  # Load balance
    λ::Vector{Float64}  # Price curve
end

function modelindices(data::SimpleShortTermData,horizon::Horizon)
    hours = collect(1:nhours(horizon))
    plants = data.hydrodata.plants
    if isempty(plants)
        error("No plants in data")
    end
    return SimpleShortTermIndices(hours, plants)
end

Define the required model indices
```
HydroModels.jl - Simple Example

```julia
struct SimpleShortTermIndices <: AbstractModelIndices
    hours::Vector{Int}
    plants::Vector{Symbol}
end

struct SimpleShortTermData <: AbstractModelData
    hydrodata::HydroPlantCollection{Float64,2}
    D::Vector{Float64}  # Load balance
    λ::Vector{Float64}  # Price curve
end

function modelindices(data::SimpleShortTermData,horizon::Horizon)
    hours = collect(1:nhours(horizon))
    plants = data.hydrodata.plants
    if isempty(plants)
        error("No plants in data")
    end
    return SimpleShortTermIndices(hours, plants)
end
```

Define data structure that should be available in the model
**HydroModels.jl - Simple Example**

```julia
struct SimpleShortTermIndices <: AbstractModelIndices
    hours::Vector{Int}
    plants::Vector{Symbol}
end

struct SimpleShortTermData <: AbstractModelData
    hydrodata::HydroPlantCollection{Float64,2}
    D::Vector{Float64}  # Load balance
    λ::Vector{Float64}  # Price curve
end

function modelindices(data::SimpleShortTermData,horizon::Horizon)
    hours = collect(1:nhours(horizon))
    plants = data.hydrodata.plants
    if isempty(plants)
        error("No plants in data")
    end
    return SimpleShortTermIndices(hours, plants)
end
```

Create model indices based on given data and time horizon
@hydromodel Deterministic SimpleShortTerm = begin

    ... 

    hours = indices.hours
    plants = indices.plants
    ...

    hdata = data.hydrodata
    D = data.D
    λ = data.λ
    ...

    @variable(model, H[t = hours] >= 0) # Production each hour
    ...

    @expression(model, value_of_stored_water,
        0.98*mean(λ)*sum(M[p,24]*sum(hdata[i].μ[1]
            for i = hdata.Qd[p])
            for p = plants))

    @objective(model, Max, net_profit + value_of_stored_water)
    ...

    @constraint(model, load_constraint[t = hours],
        H[t] + Hp[t] - Hs[t] == D[t])
    ...

end
simple_model = SimpleShortTermModel(Day(), data)

Deterministic Hydro Power Model: Simple Short Term
  including 5 power stations
  over a 24 hour horizon (1 day)

Not yet planned
simple_model = SimpleShortTermModel(Day(), data)

Deterministic Hydro Power Model: Simple Short Term
including 5 power stations
over a 24 hour horizon (1 day)

Not yet planned

plan!(simple_model, optimsolver = CbcSolver())

Deterministic Hydro Power Model: Simple Short Term
including 5 power stations
over a 24 hour horizon (1 day)

Optimally planned
reinitialize!(simple_model,Week(),data)

Deterministic Hydro Power Model : Simple Short Term including 5 power stations over a 168 hour horizon (1 week)

Not yet planned
reinitialize!(simple_model, Week(), data)

Deterministic Hydro Power Model : Simple Short Term
    including 5 power stations
    over a 168 hour horizon (1 week)

Not yet planned

plan!(simple_model, optimsolver = CbcSolver())

Deterministic Hydro Power Model : Simple Short Term
    including 5 power stations
    over a 168 hour horizon (1 week)

Optimally planned
HydroModels.jl - Day-Ahead Model

- HydroModels.jl model implemented using StochasticPrograms.jl
HydroModels.jl - Day-Ahead Model

- HydroModels.jl model implemented using StochasticPrograms.jl
- Determine optimal order strategies on day-ahead electricity markets
HydroModels.jl - Day-Ahead Model

- HydroModels.jl model implemented using StochasticPrograms.jl
- Determine optimal order strategies on day-ahead electricity markets
- Small benchmark
  - 257 Swedish power stations
  - 20 Price curve scenarios from the NordPool market
  - 748042 variables and 376700 constraints in the extended form
HydroModels.jl - Day-Ahead Model

- HydroModels.jl model implemented using StochasticPrograms.jl
- Determine optimal order strategies on day-ahead electricity markets
- Small benchmark
  - 257 Swedish power stations
  - 20 Price curve scenarios from the NordPool market
  - 748042 variables and 376700 constraints in the extended form
- Results
  - Gurobi on extended form: 58.2 seconds (+ 9.2s for DEP generation)
  - Distributed L-shaped: 31.5 seconds
  - Distributed L-shaped with tuned trust-region: 26.7 seconds
Final Remarks - Outlook on Future Work

- StochasticPrograms.jl
  - Sampling
  - Multistage models
  - Progressive hedging solver

- HydroModels.jl
  - Implement more models of hydropower operations

- LShapedSolvers.jl
  - Algorithmic improvements
  - Hardware acceleration
  - Support integer problems
Final Remarks - Outlook on Future Work

- StochasticPrograms.jl
  - Sampling
  - Multistage models
  - Progressive hedging solver

- HydroModels.jl
  - Implement more models of hydropower operations
Final Remarks - Outlook on Future Work

- **StochasticPrograms.jl**
  - Sampling
  - Multistage models
  - Progressive hedging solver

- **HydroModels.jl**
  - Implement more models of hydropower operations

- **LShapedSolvers.jl**
  - Algorithmic improvements
  - Hardware acceleration
  - Support integer problems
Final Remarks - Summary

- Stochastic programming for hydropower operations in Julia
  - StochasticPrograms.jl
  - LShapedSolvers.jl
  - HydroModels.jl

Disclaimer: Not updated for MathOptInterface and JuMP 0.19

All packages are available on Github:
- https://github.com/martinbiel/StochasticPrograms.jl
- https://github.com/martinbiel/LShapedSolvers.jl
- https://github.com/martinbiel/HydroModels.jl

Feedback appreciated!

Martin Biel (KTH)
Final Remarks - Summary

- Stochastic programming for hydropower operations in Julia
  - StochasticPrograms.jl
  - LShapedSolvers.jl
  - HydroModels.jl

- Software innovations
  - Deferred model creation
  - Data injection

Disclaimer: Not updated for MathOptInterface and JuMP 0.19

All packages are available on Github:
- https://github.com/martinbiel/StochasticPrograms.jl
- https://github.com/martinbiel/LShapedSolvers.jl
- https://github.com/martinbiel/HydroModels.jl

Feedback appreciated!

Martin Biel (KTH)
Final Remarks - Summary

- Stochastic programming for hydropower operations in Julia
  - StochasticPrograms.jl
  - LShapedSolvers.jl
  - HydroModels.jl

- Software innovations
  - Deferred model creation
  - Data injection

- Disclaimer: Not updated for MathOptInterface and JuMP 0.19
Final Remarks - Summary

- Stochastic programming for hydropower operations in Julia
  - StochasticPrograms.jl
  - LShapedSolvers.jl
  - HydroModels.jl

- Software innovations
  - Deferred model creation
  - Data injection

- Disclaimer: Not updated for MathOptInterface and JuMP 0.19

- All packages are available on Github:
  - https://github.com/martinbiel/StochasticPrograms.jl
  - https://github.com/martinbiel/LShapedSolvers.jl
  - https://github.com/martinbiel/HydroModels.jl

Feedback appreciated!