Solving optimization problems

Przemysław Szufel
https://szufel.pl/

Preparation of this workshop has been supported by the Polish National Agency for Academic Exchange under the Strategic Partnerships programme, grant number BPI/PST/2021/1/00069/U/00001.

SGH
Warsaw School of Economics

NAWA POLISH NATIONAL AGENCY FOR ACADEMIC EXCHANGE

Basics…

Linear optimization

```julia
using JuMP, HiGHS
m = Model(optimizer_with_attributes(HiGHS.Optimizer))
@variable(m, x₁ >= 0)
@variable(m, x₂ >= 0)
@objective(m, Min, 50x₁ + 70x₂)
@constraint(m, 200x₁ + 2000x₂ >= 9000)
@constraint(m, 100x₁ +   30x₂ >=  300)
@constraint(m, 9x₁   +   11x₂ >=   60)
optimize!(m)
JuMP.value.([x₁,x₂])
```

Note – how to type indexes in Julia

- julia> x
- julia> x_
- julia> x_1
- julia> x_1***<TAB>***
- julia> x_1

… and Integer programming

```
using JuMP, HiGHS
m = Model(optimizer_with_attributes(HiGHS.Optimizer))
@variable(m, x₁ >= 0, Int)
@variable(m, x₂ >= 0)
@objective(m, Min, 50x₁ + 70x₂)
@constraint(m, 200x₁ + 2000x₂ >= 9000)
@constraint(m, 100x₁ +   30x₂ >=  300)
@constraint(m, 9x₁   +   11x₂ >=   60)
optimize!(m)
```

How it works - metaprogramming

```
julia> code = Meta.parse("x=5")
:(x = 5)

julia> dump(code)
Expr
  head: Symbol =
  args: Array{Any}((2,))
    1: Symbol x
    2: Int64 5

julia> eval(code)
5

julia> x
5
```

Macros – hello world...

```
macro sayhello(name)
	return :( println("Hello, ", $name) )
end

julia> macroexpand(Main,:(@sayhello("aa")))
:((Main.println)("Hello, ", "aa"))

julia> @sayhello "world!"
Hello, world!
```

Macro @variable

```
julia> @macroexpand @variable(m, x₁ >= 0)
quote
    (JuMP.validmodel)(m, :m)
    begin
        #1###361 = begin
                let
                    #1###361 = (JuMP.constructvariable!)(m, getfield(JuMP, Symbol("#_error#107")){Tuple{Symbol,Expr}}((:m, :(x₁ >= 0))), 0, Inf, :Default, (JuMP.string)(:x₁), NaN)
                    #1###361
                end
            end
        (JuMP.registervar)(m, :x₁, #1###361)
        x₁ = #1###361
    end
end
```

Some of JuMP Solvers (over 40 as of today)

Solver	Julia Package	License	LP	SOCP	MILP	NLP	MINLP	SDP
Artelys Knitro	KNITRO.jl	Comm.				X	X	
BARON	BARON.jl	Comm.				X	X	
Bonmin	AmplNLWriter.jl CoinOptServices.jl	EPL	X		X	X	X	
Cbc	**Cbc.jl**	**EPL**			**X**			
Clp	Clp.jl	EPL	X					
Couenne	AmplNLWriter.jl CoinOptServices.jl	EPL	X		X	X	X	
CPLEX	**CPLEX.jl**	Comm.	X	X	X			
ECOS	ECOS.jl	GPL	X	X				
FICO Xpress	Xpress.jl	Comm.	X	X	X			
HiGHS	HiGHSMathProgInterface	GPL	X		X			
Gurobi	**Gurobi.jl**	Comm.	X	X	X			
Ipopt	**Ipopt.jl**	**EPL**	X			X		
MOSEK	Mosek.jl	Comm.	X	X	X	X		X
NLopt	NLopt.jl	LGPL				X		
SCS	SCS.jl	MIT	X	X				X

JuMP
Transportation of good among branches

Use case scenario

The Subway restaurant chain in Las Vegas has a total of 118 restaurants in different parts of the city.

18 restaurants have adjacent huge product warehouses that keep ingredients cool and fresh, moreover fresh vegetables are delivered only to those warehouses (rather than to every restaurant) daily at 3am.

Subway has signed a contract with a transportation agency and is billed by the multiple of the weight of transported goods and the distance.

Knowing the amount of available stock at each warehouse and the expected demand at each restaurant (measured in kg), the company needs to decide how the goods should be distributed among warehouses.

Transportation problem statement

- Variables
  - x_{ij} – number of units transported for i-th supplier to j-th requester
  - c_{ij} – unit transportation cost between i-th supplier to j-th requester
- Cost function C

$$C = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

- Constraints:
  suppliers have maximum capacity S_i

$$\sum_{j=1}^{n} x_{ij} \le S_i$$

demand D_j must be met

$$\sum_{i=1}^{m} x_{ij} \ge D_j$$

Implementation in JuMP

```
m = Model(optimizer_with_attributes(HiGHS.Optimizer));
@variable(m, x[i=1:S, j=1:D])
@objective(m, Min, sum( x[i, j]*distance_mx[i, j] for i=1:S, j=1:D))
@constraint(m, x .>= 0)
for j=1:D
   @constraint(m, sum( x[i, j] for i=1:S) >= demand[j]   )
end
for i=1:S
   @constraint(m, sum( x[i, j] for j=1:D) <= supply[i]   )
end
optimize!(m)
termination_status(m)
```

JuMP
Travelling salesman problem

Use case scenario

The Subway restaurant chain in Las Vegas has a total of 118 restaurants in different parts of the city.

Company’s manager plans to visit all restaurants during a single day.

What is the optimal order that restaurants should be visited?

Traveling salesman problem (TSP)

- Variables:
  - c_{ft} – cost of travel from "f" to "t"
  - x_{ft} – binary variable indicating 1 when agent travels from "f" to "t"

$$\text{Min} \sum_{f=1}^{N} \sum_{t=1}^{N} c_{ft} x_{ft}$$

The mathematical problem framing based on: Julia Programming for Operations Research A Primer on Computing, Changhyun Kwon, 2018, see: http://opensourc.es/blog/mip-tsp

TSP

$$\text{Min} \sum_{f=1}^{N} \sum_{t=1}^{N} c_{ft} x_{ft}$$

Variables:

- c_{ft} – cost of travel from "f" to "t"
- x_{ft} – binary variable indicating 1 when agent travels from "f" to "t"

Each city visited once

$$\sum_{t=1}^{N} x_{ft} = 1 \quad \forall f \in \{1, \ldots, N\}$$

$$\sum_{f=1}^{N} x_{ft} = 1 \quad \forall t \in \{1, \ldots, N\}$$

City cannot visit itself

$$x_{ff} = 0 \quad \forall f \in \{1, \ldots, N\}$$

Avoid two-city cycles

$$x_{ft} + x_{tf} <= 1 \quad \forall f, t \in \{1, \ldots, N\}$$

Other cycles: /dynamically add a constraint whenever a cycle occurs/

For more details see: http://opensourc.es/blog/mip-tsp

JuMP implementation

```
m = Model(optimizer_with_attributes(HiGHS.Optimizer));
@variable(m, x[f=1:N, t=1:N], Bin)
@objective(m, Min, sum( x[i, j]*distance_mx[i,j] for i=1:N,j=1:N))
@constraint(m, notself[i=1:N], x[i, i] == 0)
@constraint(m, oneout[i=1:N], sum(x[i, 1:N]) == 1)
@constraint(m, onein[j=1:N], sum(x[1:N, j]) == 1)
for f=1:N, t=1:N
    @constraint(m, x[f, t]+x[t, f] <= 1)
end
```

Getting a cycle

```
function getcycle(m, N)
    x_val = getvalue(x)
    cycle_idx = Vector{Int}()
    push!(cycle_idx, 1)
    while true
        v, idx = findmax(x_val[cycle_idx[end], 1:N])
        if idx == cycle_idx[1]
            break
        else
            push!(cycle_idx, idx)
        end
    end
    cycle_idx
end
```

Adding a constraint…

```
function solved(m, cycle_idx, N)
    println("cycle_idx: ", cycle_idx)
    println("Length: ", length(cycle_idx))
    if length(cycle_idx) < N
        cc = @constraint(m, sum(x[cycle_idx,cycle_idx])
           <= length(cycle_idx)-1)
        println("added a constraint")
        return false
    end
    return true
end
```

Iterating over the model

```
while true
    status = solve(m)
    println(status)
    cycle_idx = getcycle(m, N)
    if solved(m, cycle_idx,N)
        break;
    end
end
```

Gurobi.jl

- Commercial software
- Free for academic use
- Integrates with JuMP via Gurobi.jl

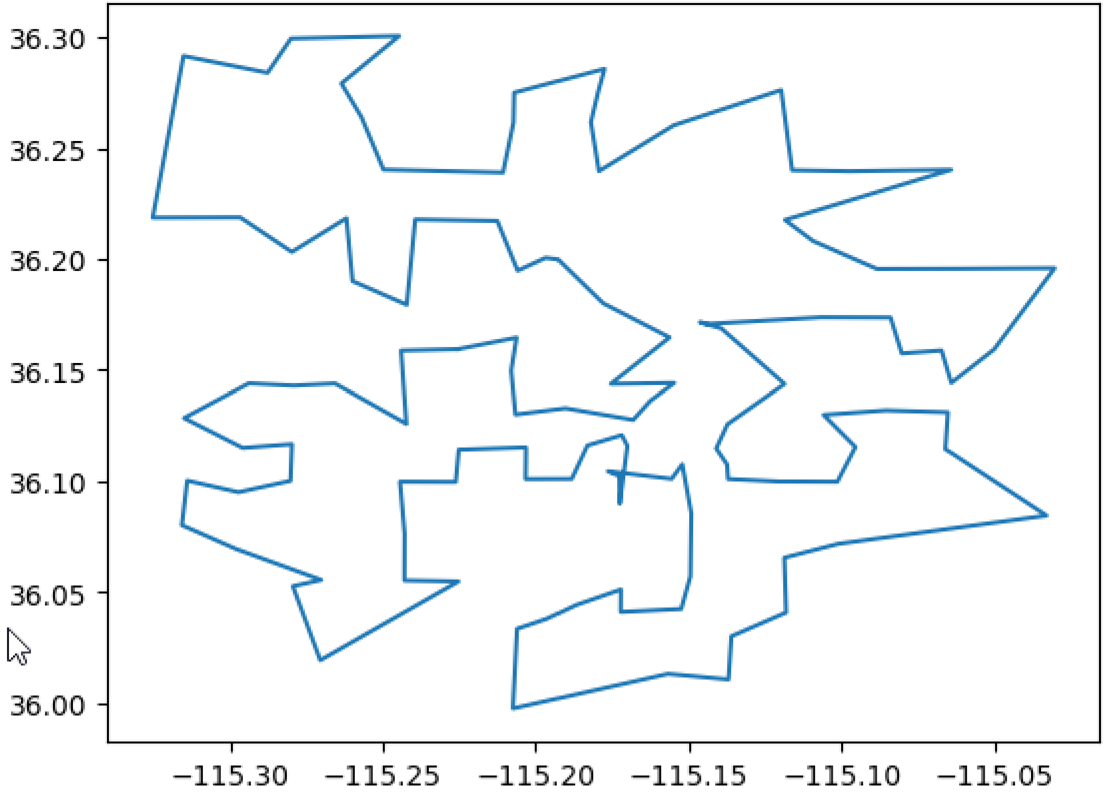
- Supports JuMP Lazy constraints (http://www.juliaopt.org/JuMP.jl/0.18/callbacks.html)

Gurobi callbacks

```
function getcycle(cb, N)
    x_val = callback_value.(Ref(cb), x)
    getcycle(x_val)
end
function callbackhandle(cb)
    cycle_idx = getcycle(cb, N)
    println("Callback! N= $N cycle_idx: ", cycle_idx)
    println("Length: ", length(cycle_idx))
    if length(cycle_idx) < N
        con = @build_constraint(sum(x[cycle_idx,cycle_idx]) <= length(cycle_idx)-1)
        MOI.submit(m, MOI.LazyConstraint(cb), con)
        println("added a lazy constraint")
    end
end
MOI.set(m, MOI.LazyConstraintCallback(), callbackhandle)
```

TravelingSalesmanHeuristics.jl

```
using TravelingSalesmanHeuristics
sol = TravelingSalesmanHeuristics.solve_tsp(
distance_mx,quality_factor =100)
```



More info:
http://evanfields.github.io/TravelingSalesmanHeuristics.jl/latest/heuristics.html

JuMP
Non-Linear Programming

Simple scenario

Estimate parameters of a quadratic form

$$\boldsymbol{y}(\boldsymbol{x}_i) = \boldsymbol{x}_i^T \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \boldsymbol{x}_i, \text{ where } \boldsymbol{x}_i = \begin{bmatrix} x_i^1 \\ x_i^2 \end{bmatrix}$$

for a vector of observed values ***y*** to minimize the observed error function

$$\sum_{i=1}^{N} (y(\boldsymbol{x}_i) - y_i)^2$$

Nonlinear optimization Julia

```
m = Model(optimizer_with_attributes(Ipopt.Optimizer));

@variable(m, aa[1:2,1:2])

function errs(aa)
   sum((y .- (x * aa ) .* x * [1;1]) .^ 2)
end

@objective(m, Min, errs(aa))

optimize!(m)
```

Use case scenario

(source: Hart et al, Pyomo-optimization modeling in python, 2017)

Simulate dynamics of disease outbreak in a small community of 300 individuals (e.g. children at school)

Three possible states of a patient:

- susceptible (S)
- infected (I)
- recovered (R)

<u>Infection spread model :</u>

- N – population size
- α, β – model parameters

$$I_i = \frac{\beta I_{i-1}^{\alpha} S_{i-1}}{N}$$

$$S_i = S_{i-1} - I_i$$

Optimization problem for finding parameters α and β

S - susceptible

I - infected

N – population size

α, β – model parameters

SI - time indices {1,2,3,...}

C_i - known input (the actual number of infected patients)

$$\min \sum_{i \in SI} \left(\varepsilon_i^I\right)^2$$

$$I_i = \frac{\beta I_{i-1}^{\alpha} S_{i-1}}{N} \quad \forall \ i \in SI \setminus \{1\}$$

$$S_i = S_{i-1} - I_i \ \forall \ i \ \in SI \setminus \{1\}$$

$$C_i = I_i + \varepsilon_i^I$$

$$0 \leq I_i, \ Si \leq N$$

$$0.5 \leq \beta \leq 70$$

$$0.5 \leq \alpha \leq 1.5$$

Model implementation in JuMP

- Input data (disease dynamics)

```
obs_cases = vcat(1,2,4,8,15,27,44,58,55,32,12,3,1,zeros(13))
```

Full model specification in JuMP

```
m = Model(optimizer_with_attributes(Ipopt.Optimizer));
@variable(m, 0.5 <= α <= 1.5)
@variable(m, 0.05 <= β <= 70)
@variable(m, 0 <= I_[1:SI_max] <= N)
@variable(m, 0 <= S[1:SI_max]  <= N)
@variable(m, ε[1:SI_max])
@constraint(m, ε .== I_ .- obs_cases  )
@constraint(m, I_[1] == 1)
for i=2:SI_max
   @NLconstraint(m, I_[i] == β*(I_[i-1]^α)*S[i-1]/N)
end
@constraint(m, S[1] == N)
for i=2:SI_max
   @constraint(m, S[i] == S[i-1]-I_[i])
end
@NLobjective(m, Min, sum(ε[i]^2 for i in 1:SI_max))
```

JuMP
Non-Linear Programming
for estimation of model parameters

Simple scenario

Estimate parameters of a quadratic form

$$\boldsymbol{y}(\boldsymbol{x}_i) = \boldsymbol{x}_i^T \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \boldsymbol{x}_i, \text{ where } \boldsymbol{x}_i = \begin{bmatrix} x_i^1 \\ x_i^2 \end{bmatrix}$$

for a vector of observed values ***y*** to minimize the observed error function

$$\sum_{i=1}^{N} (y(\boldsymbol{x}_i) - y_i)^2$$

Nonlinear optimization Julia

```
m = Model(optimizer_with_attributes(Ipopt.Optimizer));

@variable(m, aa[1:2,1:2])

function errs(aa)
   sum((y .- (x * aa ) .* x * [1;1]) .^ 2)
end

@objective(m, Min, errs(aa))

optimize!(m)
```

Use case scenario

(source: Hart et al, Pyomo-optimization modeling in python, 2017)

Simulate dynamics of disease outbreak in a small community of 300 individuals (e.g. children at school)

Three possible states of a patient:

- susceptible (S)
- infected (I)
- recovered (R)

<u>Infection spread model :</u>

- N – population size
- α, β – model parameters

$$I_i = \frac{\beta I_{i-1}^{\alpha} S_{i-1}}{N}$$

$$S_i = S_{i-1} - I_i$$

Optimization problem for finding parameters α and β

S - susceptible

I - infected

N – population size

α, β – model parameters

SI - time indices {1,2,3,...}

C_i - known input (the actual number of infected patients)

$$\min \sum_{i \in SI} \left(\varepsilon_i^I\right)^2$$

$$I_i = \frac{\beta I_{i-1}^{\alpha} S_{i-1}}{N} \ \forall \ i \in SI \setminus \{1\}$$

$$S_i = S_{i-1} - I_i \ \forall \ i \ \in SI \setminus \{1\}$$

$$C_i = I_i + \varepsilon_i^I$$

$$0 \leq I_i, \ Si \leq N$$

$$0.5 \leq \beta \leq 70$$

$$0.5 \leq \alpha \leq 1.5$$

Model implementation in JuMP

- Input data (disease dynamics)

```
obs_cases = vcat(1,2,4,8,15,27,44,58,55,32,12,3,1,zeros(13))
```

Full model specification in JuMP

```
m = Model(optimizer_with_attributes(Ipopt.Optimizer));
@variable(m, 0.5 <= α <= 1.5)
@variable(m, 0.05 <= β <= 70)
@variable(m, 0 <= I_[1:SI_max] <= N)
@variable(m, 0 <= S[1:SI_max]  <= N)
@variable(m, ε[1:SI_max])
@constraint(m, ε .== I_ .- obs_cases  )
@constraint(m, I_[1] == 1)
for i=2:SI_max
   @NLconstraint(m, I_[i] == β*(I_[i-1]^α)*S[i-1]/N)
end
@constraint(m, S[1] == N)
for i=2:SI_max
   @constraint(m, S[i] == S[i-1]-I_[i])
end
@NLobjective(m, Min, sum(ε[i]^2 for i in 1:SI_max))
```

JuMP
Multi-criteria optimization
for stock portfolio optimization

[additional material]

Stock portfolio optimization

- Estimate the weights vector

$$\vec{x} = \left[x_1\ x_2 \cdots x_n\right]^T$$

  where x_i represents the share of asset i in a portfolio: $\vec{1}^T\vec{x} = 1$

- maximize the expected return $x_p = \vec{p}^T\vec{x} = \sum_{i=1}^{n} x_i p_i$
- minimize the risk

$$\sigma_p^{\ 2} = \vec{x}^T V \vec{x} = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j \sigma_{i,j}$$

for the variance-covariance matrix:

$$V = \begin{pmatrix} \sigma_{11} & \cdots & \sigma_{1n} \\ \vdots & \ddots & \vdots \\ \sigma_{n1} & \cdots & \sigma_{nn} \end{pmatrix}$$

Source: Duan Y. C. "A multi-objective approach to portfolio optimization." *Rose-Hulman Undergraduate Mathematics Journal* 8.1 (2007): 12.

Possible approaches

- Maximize the expected return, disregarding risk
- Minimize the expected risk, disregarding return
- Maximize the return for a given level of risk
- Minimize the risk for a given level of return

- **Maximize the risk** AND **minimize the return**
  ➔ **multi-criteria optimization**

Stock price data

- Top 10 Fortune 500 companies
- 3 years
- Daily opening and closing prices

- Expected return
  - average daily rate of return (for simplicity calculated as a difference between opening and closing price)
- Risk
  - calculate variance-covariance matrix for daily returns

Julia implementation – data processing

```
prices = CSV.read("10_stocks_3yr.csv", allowmissing=:none)
prices.rateOfRet =
     (prices.close-prices.open) ./ prices.open
dats = unstack(prices, :date, :Name, :rateOfRet)
const avg_rets = colwise(mean, dats[2:end])
const cov_mx = cov(Matrix(dats[2:end]))
```

Julia MultiJuMP model

```
m = multi_model(Ipopt.Optimizer)
@variable(m, 0 <= x[i=1:10] <= 1)
@constraint(m,sum(x) == 1)
@variable(m, risk)
@constraint(m, risk == x'*cov_mx*x)
@variable(m, rets)
@constraint(m, rets == avg_rets' * x)
@NLexpression(m, f_risk, risk)
@NLexpression(m, f_rets, rets)
```

Solving the model

```
iv1 =  fill(0.1, 10) # Initial guess
obj1 = SingleObjective(f_risk, sense = MOI.MIN_SENSE,
                    iv = Dict{String,Any}("x[$i]" => iv1[i] for i in 1:length(iv1)))
obj2 = SingleObjective(f_rets, sense = MOI.MAX_SENSE)
md = get_multidata(m)
md.objectives = [obj1, obj2]
md.pointsperdim = 20

optimize!(m, method = NBI(false))
```

Plotting the Pareto-frontier

```
Plots.pyplot()
Plots.plot(md)
```