JuMP
Non-Linear Programming

Przemysław Szufel
https://szufel.pl/

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(solver = Ipopt.IpoptSolver());
@variable(m, aa[1:2,1:2])

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

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

status = solve(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(solver = Ipopt.IpoptSolver());
@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))
```