JuMP
Multi-criteria optimization for stock portfolio optimization

Przemysław Szufel
https://szufel.pl/

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 = MultiModel(solver = IpoptSolver())
@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 = :Min,
     iv = Dict{Symbol,Any}(:x => iv1))
obj2 = SingleObjective(f_rets, sense = :Max)
md = getMultiData(m)
md.objectives = [obj1, obj2]
md.pointsperdim = 20

open("solver_trace.txt", "w") do io
    redirect_stdout(io) do
        solve(m, method = :NBI)
    end
end
```

Plotting the Pareto-frontier

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