| 15.094J/1.142J Robust Modeling, Optimization and Computation | **Homework 3** | Exercise 3.1 **By: Hanzhang Qin** |
|---|---|---|

**Deterministic formulation**

**(a)** In each time period $t \in \mathcal{T}$, the service provider needs to make three types of decisions:

(i) How many units $z_{t,i,s}$ we need to expand resources $R_{i,s}$, $\forall\ i \in \mathcal{I}$, $\forall\ s \in \mathcal{S}$.

(ii) For each job $j \in \mathcal{J}$, decide how many units $y_{t,j,s',s}$ we need to allocate from server $s'$ to server $s$ ($\forall\ s, s' \in \mathcal{S}$).

(iii) Based on the job arrival $\{d_{t,j}, j \in \mathcal{J}\}$, the resource capacities and the reallocation decisions, decide on each server $s \in \mathcal{S}$, how many units $x_{t,j,s}$ of job $j$ should be produced.

Note only $x_{t,j,s}$ and $y_{t,j,s',s}$ affect the queue lengths, and we have:

$$q_{t,j,s} = q_{t-1,j,s} - x_{t,j,s} + \sum_{s' \neq s \in \mathcal{S}} y_{t,j,s',s} - \sum_{s' \neq s \in \mathcal{S}} y_{t,j,s,s'}.$$

**(b)** For simplicity, we shift and let the planning horizon starts at time 0 and assume assume the initial queue length ($q_{0,j}$) is known for all $j$ and denote the costs in last period $c_j^o$ as $c_{j,T}$. The deterministic linear optimization formulation is stated as follows.

$$\begin{aligned}
\min\ & \sum_{t=0}^{T}\sum_{j=1}^{J} c_{t,j} q_{t,j} + \sum_{t=0}^{T-1}\sum_{i=1}^{I}\sum_{s=1}^{S} c^e_{t,i,s} z_{t,i,s} + \sum_{t=0}^{T-1}\sum_{j=1}^{J}\sum_{s \neq s' \in \mathcal{S}}^{S} c^r_j y_{t,j,s,s'}, \\
\text{s.t. } & q_{t,j,s} = q_{t-1,j,s} - x_{t,j,s} + \sum_{s' \neq s \in \mathcal{S}} y_{t,j,s',s} - \sum_{s' \neq s \in \mathcal{S}} y_{t,j,s,s'}, \forall\ t = 0, \ldots, T, \forall\ j = 1, \ldots, J, \forall\ s = 1, \ldots, S, \\
& \sum_{j=1}^{J} r_{i,j,s} x_{t,j,s} \leq R_{i,s} + z_{t,i,s}, \forall\ t = 0, \ldots, T-1, \forall\ i = 1, \ldots, I, \forall\ s = 1, \ldots, S, \\
& \sum_{t=0}^{m} x_{t,j,s} \leq \sum_{t=0}^{m} d_{t,j}, \forall\ j = 1, \ldots, J, \forall\ m = 1, \ldots, T-1, \\
& y_{t,j,s,s'} \leq q_{t,j,s}, \forall\ t = 0, \ldots, T, \forall\ j = 1, \ldots, J, \forall\ s \neq s' = 1, \ldots, S, \\
& x_{t,j,s}, y_{t,j,s,s'}, z_{t,i,s} \geq 0, \forall\ i, j, t, s \neq s'.
\end{aligned}$$

**Robust formulation**

**(a)** The simplest way to formulate the uncertainty set might be assuming the historical data are i.i.d with respect to time (but still dependent on jobs) and we use the central limit theorem to construct the uncertainty set as follows:

$$\mathcal{U}^{\text{CLT}} = \left\{ \forall\ j \in \mathcal{J} : \left| \sum_{t=0}^{T-1} d_{t,j} - \mu_j \right| \leq \Gamma \sigma_j \sqrt{T} \right\},$$

where we denote $\mu_j$ and $\sigma_j$ the sample mean and sample standard deviation of the $N$ samples drawn from the distribution $d_{t,j}$, $\Gamma$ is a positive constant.

However, it is often natural that time correlation exists in a finite horizon optimization problem. That is, $\forall\ j \in \mathcal{J}, d_{t,j}$ might depend on $d_{t-1,j}, \ldots, d_{0,j}$. One way to model this kind of uncertainty is to construct

$$\mathcal{U}^{\text{Cor}} = \left\{ \forall\ j \in \mathcal{J} : \boldsymbol{d}_j = \boldsymbol{A}_j \boldsymbol{y}_j + \boldsymbol{\epsilon}_j, \left| \sum_{l=1}^{m} y_{t,l} - m\mu_j \right| \leq \Gamma \sigma_j^y \sqrt{m}, \left| \sum_{t=0}^{T-1} \epsilon_t \right| \leq \Gamma \sigma_j^\epsilon \right\},$$

where $\mathbf{A}_j$ is an $t \times m$ matrix.

**(b)** To make the robust problem tractable, the most natural solution is to only consider the affine adaptable solutions. To be specific, given the uncertainty set $\mathcal{U}^{\text{CLT}}$ or $\mathcal{U}^{\text{Cor}}$, we assume decision variables $x_{t,j,s}, y_{t,j,s,s'}, z_{t,i,s}$ are affine functions of $d_{t,j}$:

$$x_{t,j,s} = x_{t,j,s}^0 + \sum_{\tau=1}^{T} \sum_{j=1}^{J} x_{t,j,s}^\tau d_{\tau-1,j}, y_{t,j,s,s'} = x_{t,j,s}^0 + \sum_{\tau=1}^{T} \sum_{j=1}^{J} y_{t,j,s,s'}^\tau d_{\tau-1,j}, z_{t,j,s} = x_{t,i,s}^0 + \sum_{\tau=1}^{T} \sum_{j=1}^{J} z_{t,i,s}^\tau d_{\tau-1,j}.$$

So, we again end up with a linear optimization problem, which is polynomial-time solvable.

**(c)** In general, the proposed robust uncertainty set might provide much too conservative solutions for multi-period optimization problems, compared to the stochastic programming formulations. However, the robust problem is much easier to solve than the stochastic counterparts especially in high dimensions.

Specifically, in reality $\mathcal{U}^{\text{CLT}}$ is easier to construct (we only need to compute sample mean and standard deviation) but it fails to capture the prospective time correlation. On the contrary, $\mathcal{U}^{\text{Cor}}$ captures this effect but it might be hard in reality to find appropriate $\boldsymbol{A}_j$.

**(d)** It is possible for the demand to change in the future. $\mathcal{U}^{\text{CLT}}$ clearly is unable to capture this change, $\mathcal{U}^{\text{Cor}}$ might be able to capture this change as long as this change is seasonal and $\boldsymbol{A}_j$ corresponds to this kind of seasonality. Or, a more specific approach is that we can consider fit an autocorrelated model using the past samples, say, $\forall\ j \in \mathcal{J}$,

$$d_{t,j} = a_0 + \sum_{l=1}^{q} d_{t-l,j} + \epsilon_{t,j},$$

and the uncertainty set is given by

$$\mathcal{U}^{\text{AR}(q)} = \left\{ \forall\ t \in \mathcal{T}, \forall\ j \in \mathcal{J}, d_{t,j} = a_0 + \sum_{l=1}^{q} d_{t-l,j} + \epsilon_{t,j}, \left| \sum_{t=1}^{T} \epsilon_{t,j} \right| \leq \Gamma \right\}.$$

**Model where facilities already constructed**

**When demands are known** Obviously, in this situation, no reallocating is needed (as long as $t_{ij} > 0$ which is always true in practice) since we have perfect information of the demands. The linear optimization problem is formulated as

$$\begin{aligned} \max\ & p\sum_{i=1}^{n} s_i - \sum_{i=1}^{n} x_i \\ \text{s.t. } & s_i \le d_i, \forall\ i = 1, \dots, n \\ & s_i \le x_i, \forall\ i = 1, \dots, n, \\ & x_i, s_i \ge 0, \forall\ i = 1, \dots, n. \end{aligned}$$

And it is easy to note the optimal solution is to set $x_i = d_i$ for all $i$.

**When demands are uncertain** Since after demands are realized, we need to reallocate stock so this is a two-stage adaptive optimization problem. Specifically, under any uncertainty set $\mathcal{U}$, the problem can be formulated as

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{s}}\ & p\sum_{i=1}^{n} s_i - \sum_{i=1}^{n} x_i - \max_{\boldsymbol{d}\in\mathcal{U}} \min_{\boldsymbol{y}(\boldsymbol{d})} \sum_{1\le i\ne j\le n} t_{ij} y_{ij}(\boldsymbol{d}) \\ \text{s.t. } & s_i \le d_i, \forall\ i = 1, \dots, n, \forall\ \boldsymbol{d} \in \mathcal{U}, \\ & s_i \le x_i - \sum_{j\ne i} y_{ij}(\boldsymbol{d}) + \sum_{j\ne i} y_{ji}(\boldsymbol{d}), \forall\ i = 1, \dots, n, \forall\ \boldsymbol{d} \in \mathcal{U}, \\ & x_i, s_i \ge 0, \forall\ i = 1, \dots, n, \\ & y_{ij}(\boldsymbol{d}) \ge 0, \forall\ i \ne j = 1, \dots, n. \end{aligned} \tag{AO}$$

To transform the problem as a robust optimization problem, we simply drop the dependency of $\boldsymbol{y}$ on $\boldsymbol{d}$ and have

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{s},-\boldsymbol{y}}\ & p\sum_{i=1}^{n} s_i - \sum_{i=1}^{n} x_i - \sum_{1\le i\ne j\le n} t_{ij} y_{ij} \\ \text{s.t. } & s_i \le d_i, \forall\ i = 1, \dots, n, \forall\ \boldsymbol{d} \in \mathcal{U}, \\ & s_i \le x_i - \sum_{j\ne i} y_{ij} + \sum_{j\ne i} y_{ji}, \forall\ i = 1, \dots, n, \forall\ \boldsymbol{d} \in \mathcal{U}, \\ & x_i, s_i \ge 0, \forall\ i = 1, \dots, n, \\ & y_{ij} \ge 0, \forall\ i \ne j = 1, \dots, n. \end{aligned} \tag{RO}$$

Finally, for the affinely adjustable optimization (AAO) problem, we just need to specify $y_{ij}(\boldsymbol{d})$ in (AO) in the form of an affine function: $y_{ij}(\boldsymbol{d}) = a_{ij}^0 + \sum_{l=1}^{n} a_{ij}^l d_l$.

**Computational results** We model the RO and AAO problems using JuMPeR and evaluate their performance based on randomly generated $\boldsymbol{d}$ and the first stage decisions. For each $n$, we test 5 replications and taking the average of the measure $\rho = \frac{\text{Obj(AAO)-Obj(RO)}}{Obj(AAO)}$. From the table, we see as $n$ goes large, $\rho$ tends to be quite small (the difference between RO and AAO is negligible).

Table 1: $\rho$ for AAO and RO where facilities already constructed.

| n | 5 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|
| $\rho$ | .147 | .115 | .086 | .002 | $< 1e^{-3}$ |

**Model where facilities not yet constructed**

**When demands are known** The only change is we need to pay $c$ if we want to build facilities at location $i$ otherwise $x_i$ can only be zero. For this reason, we need to introduce binary variables $b_i = 1$ means we build facilities at location $i$ and $b_i = 0$ otherwise. Then, we have the following MILO formulation: ($M > 0$ is a big constant)

$$\begin{aligned}
\max\ & p\sum_{i=1}^{n} s_i - \sum_{i=1}^{n} x_i - c\sum_{i=1}^{n} b_i \\
\text{s.t. } & s_i \le d_i, \forall\ i = 1, \dots, n \\
& s_i \le x_i, \forall\ i = 1, \dots, n, \\
& x_i \le M b_i, \forall\ i = 1, \dots, n, \\
& x_i, s_i \ge 0, b_i \in \{0, 1\}, \forall\ i = 1, \dots, n.
\end{aligned}$$

**When demands are uncertain** The way to introduce uncertainty is the same as the models where facilities are already built. For example, the general adaptive optimization formulation is stated as:

$$\begin{aligned}
\max_{\boldsymbol{x},\boldsymbol{s}}\ & p\sum_{i=1}^{n} s_i - \sum_{i=1}^{n} x_i - c\sum_{i=1}^{n} b_i - \max_{\boldsymbol{d}\in\mathcal{U}} \min_{\boldsymbol{y}(\boldsymbol{d})} \sum_{1\le i\ne j\le n} t_{ij} y_{ij}(\boldsymbol{d}) \qquad \text{(AO)} \\
\text{s.t. } & s_i \le d_i, \forall\ i = 1, \dots, n, \forall\ \boldsymbol{d} \in \mathcal{U}, \\
& s_i \le x_i - \sum_{j\ne i} y_{ij}(\boldsymbol{d}) + \sum_{j\ne i} y_{ji}(\boldsymbol{d}), \forall\ i = 1, \dots, n, \forall\ \boldsymbol{d} \in \mathcal{U}, \\
& x_i \le M b_i, \forall\ i = 1, \dots, n, \\
& x_i, s_i \ge 0, b_i \in \{0, 1\}, \forall\ i = 1, \dots, n, \\
& y_{ij}(\boldsymbol{d}) \ge 0, \forall\ i \ne j = 1, \dots, n.
\end{aligned}$$

Again RO is obtained by dropping dependence of $\boldsymbol{y}(\boldsymbol{d})$ on $\boldsymbol{d}$ and AAO is obtained by specifying $y_{ij}(\boldsymbol{d}) = a^0_{ij} + \sum_{l=1}^{n} a^l_{ij} d_l$.

**Computational results** We use the same metric $\rho$ as in the previous part, and we conduct 3 replications for each $n$. We see a similar trend that as $n$ goes large $\rho$ decreases, however this trend is not as strong as before.

Table 2: $\rho$ for AAO and RO where facilities not yet constructed.

| n | 5 | 10 | 15 | 20 |
|---|---|---|---|---|
| $\rho$ | .249 | .130 | .145 | .084 |

# Pset3_JuliaCode

April 5, 2017

In [2]:
```
using JuMP, JuMPeR, Gurobi, PyPlot, Distributions
```

In [3]:
```
function RO_haveFacilities(n,p,c,μ,σ,t)
    RO0 = RobustModel(solver=GurobiSolver(OutputFlag=0))
    @uncertain(RO0, u[1:n])
    @uncertain(RO0, d[1:n])
    @variable(RO0, x[1:n]>=0)
    @variable(RO0, s[1:n]>=0)
    @variable(RO0, y_minus[1:n,1:n]<=0)

    @constraint(RO0, norm(u,1)<=n^0.5)
    @constraint(RO0, norm(u,Inf)<=1)
    @constraint(RO0, d .== μ+σ.*u)

    for i = 1:n
        @constraint(RO0, s[i]<=d[i])
        @constraint(RO0, s[i]<=x[i]+sum(y_minus[i,j] for j=1:n)-sum(y_minus[j,i] for j=1:n))
    end

    @objective(RO0, Max, p*sum(s)-sum(x)+sum(sum(t[i,j]*y_minus[i,j] for i=1:n ) for j=1:n))
    solve(RO0)
    return getvalue(x),getvalue(y_minus)
end
```

Out[3]: RO_haveFacilities (generic function with 1 method)

In [4]:
```
function AAO_haveFacilities(n,p,c,μ,σ,t)
    AAO0 = RobustModel(solver=GurobiSolver(OutputFlag=0))
    @uncertain(AAO0, u[1:n])
    @uncertain(AAO0, d[1:n])
    @variable(AAO0, x[1:n]>=0)
    @variable(AAO0, s[1:n]>=0)
    @variable(AAO0, F)
    @adaptive(AAO0, y_minus[i=1:n,j=1:n]<=0, policy=Affine, depends_on=d[1:n])
    @constraint(AAO0, norm(u,1)<=n^0.5)
    @constraint(AAO0, norm(u,Inf)<=1)
    @constraint(AAO0, d .== μ+σ.*u)

    for i = 1:n
        @constraint(AAO0, s[i]<=d[i])
        @constraint(AAO0, s[i]<=x[i]+sum(y_minus[i,j] for j=1:n)-sum(y_minus[j,i] for j=1:n))
    end
     @constraint(AAO0, p*sum(s)-sum(x)+sum(sum(t[i,j]*y_minus[i,j] for i=1:n ) for j=1:n)>=F)
```

```
        @objective(AAO0, Max, F)
        solve(AAO0)
        return getvalue(x)
    end
```

Out[4]: AAO_haveFacilities (generic function with 1 method)

```
In [5]: function Optimal_haveFacilities(n,p,c,μ,σ,t,d,x)
            Opt0 = Model(solver=GurobiSolver(OutputFlag=0))
            @variable(Opt0, y_minus[1:n,1:n]<=0)
            @variable(Opt0, s[1:n]>=0)
            @variable(Opt0,F)
            for i = 1:n
                @constraint(Opt0, s[i]<=d[i])
                @constraint(Opt0, s[i]<=x[i]+sum(y_minus[i,j] for j=1:n)-sum(y_minus[j,i] for j=1:n))
            end
            @constraint(Opt0, p*sum(s)-sum(x)+sum(sum(t[i,j]*y_minus[i,j] for i=1:n ) for j=1:n)>=F)
            @objective(Opt0, Max, F)
            solve(Opt0)
            return getobjectivevalue(Opt0)
        end
```

Out[5]: Optimal_haveFacilities (generic function with 1 method)

```
In [6]: function ObjVal_haveFacilities(n,p,c,x,y_minus,d)
            s = zeros(n,1)
            for i = 1:n
                s[i]=min(d[i],x[i]+sum(y_minus[i,:])-sum(y_minus[:,i]))
            end
            p*sum(s)-sum(x)+sum(sum(t.*y_minus))
        end
```

Out[6]: ObjVal_haveFacilities (generic function with 1 method)

```
In [7]: for n in 5:5:25

            err = 0
            for r = 1:5
                p = 3
                c = 30
                μ = rand(Uniform(50,150),n)
                σ = 0.5 * μ

                t = zeros(n,n)
                x_cor = rand(Uniform(0,1),n)
                y_cor = rand(Uniform(0,1),n)
                for i=1:n
                    for j=1:n
                        t[i,j] = ((x_cor[i]-x_cor[j])^2 + (y_cor[i]-y_cor[j])^2)^0.5
                    end
                end
                d = []
                for i = 1:n
                    push!(d , rand(Normal(μ[i],σ[i])))
                end
```

```
            RO_x,RO_y = RO_haveFacilities(n,p,c,μ,σ,t)
            AAO_x = AAO_haveFacilities(n,p,c,μ,σ,t)
            ROObj = ObjVal_haveFacilities(n,p,c,RO_x,RO_y,d)
            AAOObj = Optimal_haveFacilities(n,p,c,μ,σ,t,d,AAO_x)
            #print("n=",n,":",(AAOObj-ROObj)/AAOObj,"\n")

        end

    end
```

```
UndefVarError: t not defined
```

```
 in ObjVal_haveFacilities(::Int64, ::Int64, ::Int64, ::Array{Float64,1}, ::Array{Float64,2}, ::A

 in macro expansion; at .\In[7]:24 [inlined]

 in anonymous at .\<missing>:?
```

In [8]:
```
function RO_noFacilities(n,p,c,μ,σ,t)
    M = 1000
    RO1 = RobustModel(solver=GurobiSolver(OutputFlag=0))
    @uncertain(RO1, u[1:n])
    @uncertain(RO1, d[1:n])
    @variable(RO1, x[1:n]>=0)
    @variable(RO1, s[1:n]>=0)
    @variable(RO1, y_minus[1:n,1:n]<=0)
    @variable(RO1, b[1:n], Bin)

    @constraint(RO1, norm(u,1)<=n^0.5)
    @constraint(RO1, norm(u,Inf)<=1)
    @constraint(RO1, d .== μ+σ.*u)

    for i = 1:n
        @constraint(RO1, s[i]<=d[i])
        @constraint(RO1, s[i]<=x[i]+sum(y_minus[i,j] for j=1:n)-sum(y_minus[j,i] for j=1:n))
        @constraint(RO1, x[i]<=M*b[i])
    end

    @objective(RO1, Max, p*sum(s)-sum(x)+sum(sum(t[i,j]*y_minus[i,j] for i=1:n ) for j=1:n)-c*su
    solve(RO1)
    return getvalue(x),getvalue(y_minus),getvalue(b)
end
```

Out[8]:
```
RO_noFacilities (generic function with 1 method)
```

In [9]:
```
function AAO_noFacilities(n,p,c,μ,σ,t)
    M = 1000
    AAO1 = RobustModel(solver=GurobiSolver(OutputFlag=0))
    @uncertain(AAO1, u[1:n])
    @uncertain(AAO1, d[1:n])
    @variable(AAO1, x[1:n]>=0)
```

```
            @variable(AAO1, s[1:n]>=0)
            @variable(AAO1, F)
            @variable(AAO1, b[1:n], Bin)
            @adaptive(AAO1, y_minus[i=1:n,j=1:n]<=0, policy=Affine, depends_on=d[1:n])
            @constraint(AAO1, norm(u,1)<=n^0.5)
            @constraint(AAO1, norm(u,Inf)<=1)
            @constraint(AAO1, d .== μ+σ.*u)

            for i = 1:n
                @constraint(AAO1, s[i]<=d[i])
                @constraint(AAO1, s[i]<=x[i]+sum(y_minus[i,j] for j=1:n)-sum(y_minus[j,i] for j=1:n))
                @constraint(AAO1, x[i]<=M*b[i])
            end
             @constraint(AAO1, -c*sum(b)+p*sum(s)-sum(x)+sum(sum(t[i,j]*y_minus[i,j] for i=1:n ) for j=

             @objective(AAO1, Max, F)
            solve(AAO1)
            return getvalue(x),getvalue(b)
        end
```

Out[9]: AAO_noFacilities (generic function with 1 method)

```
In [10]: function Optimal_noFacilities(n,p,c,μ,σ,t,d,x,b)
             Opt1 = Model(solver=GurobiSolver(OutputFlag=0))
             @variable(Opt1, y_minus[1:n,1:n]<=0)
             @variable(Opt1, s[1:n]>=0)
             @variable(Opt1,F)
             for i = 1:n
                 @constraint(Opt1, s[i]<=d[i])
                 @constraint(Opt1, s[i]<=x[i]+sum(y_minus[i,j] for j=1:n)-sum(y_minus[j,i] for j=1:n))
             end
             @constraint(Opt1, -c*sum(b)+p*sum(s)-sum(x)+sum(sum(t[i,j]*y_minus[i,j] for i=1:n ) for j=
             @objective(Opt1, Max, F)
             solve(Opt1)
             return getobjectivevalue(Opt1)
         end
```

Out[10]: Optimal_noFacilities (generic function with 1 method)

```
In [11]: function ObjVal_noFacilities(n,p,c,t,x,y_minus,d,b)
             s = zeros(n,1)
             for i = 1:n
                 s[i]=min(d[i],x[i]+sum(y_minus[i,:])-sum(y_minus[:,i]))
             end
             -c*sum(b)+p*sum(s)-sum(x)+sum(sum(t.*y_minus))
         end
```

Out[11]: ObjVal_noFacilities (generic function with 1 method)

```
In [12]: for n in 5:5:20

             err = 0
             for r = 1:1
                 p = 3
                 c = 30
```

```
            μ = rand(Uniform(50,150),n)
            σ = 0.5 * μ

            t = zeros(n,n)
            x_cor = rand(Uniform(0,1),n)
            y_cor = rand(Uniform(0,1),n)
            for i=1:n
                for j=1:n
                    t[i,j] = ((x_cor[i]-x_cor[j])^2 + (y_cor[i]-y_cor[j])^2)^0.5
                end
            end
            d = []
            for i = 1:n
                push!(d , rand(Normal(μ[i],σ[i])))
            end
            RO_x,RO_y,RO_b = RO_noFacilities(n,p,c,μ,σ,t)
            AAO_x,AOO_b = AAO_noFacilities(n,p,c,μ,σ,t)
            ROObj = ObjVal_noFacilities(n,p,c,t,RO_x,RO_y,d,RO_b)
            AAOObj = Optimal_noFacilities(n,p,c,μ,σ,t,d,AAO_x,AOO_b)
            #print("n=",n,":",(AAOObj-ROObj)/AAOObj,"\n")

        end

    end
```

```
WARNING: Not solved to optimality, status: Infeasible
```

In [ ]:

In [ ]:

In [ ]: