Lab 7: Parking Garage Case Study

Andres Calvo (ac228)

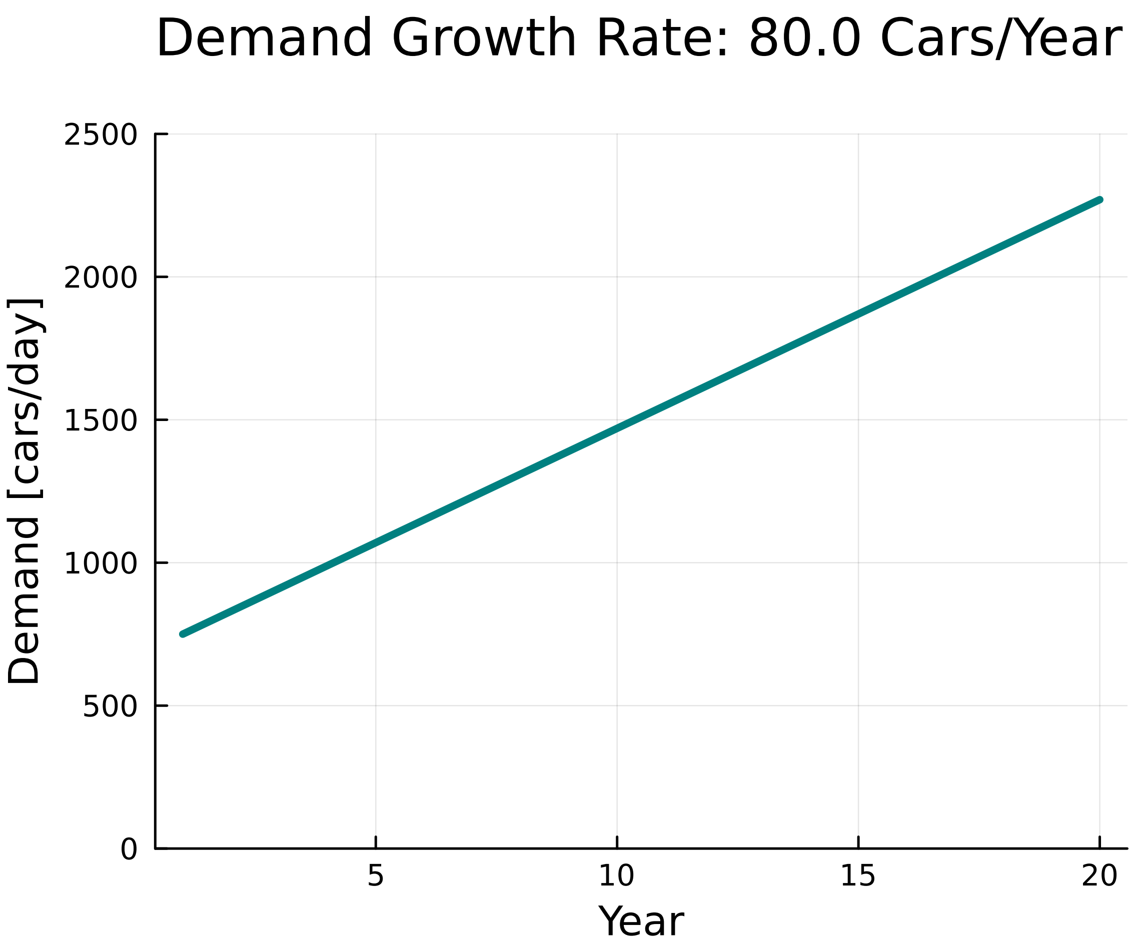
Wed., Mar. 27

```julia
using Revise
using ParkingGarage

using Plots
using Distributions

Plots.default(; margin=5Plots.mm)
```

1 Deterministic analysis

The deterministic analysis uses a **linear** demand model and constant discount rate and time frame. The following is the based demand growth.

```julia
let
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.12)
    years = 1:(sow.n_years)
    demand = [
        ParkingGarage.calculate_demand(year, sow.demand_growth_rate) for year in years
    ]
    plot(
        years,
        demand;
        ylabel = "Demand [cars/day]",
        ylims = [0,2500],
        xlabel = "Year",
        legend = false,
        title = "Demand Growth Rate: $(sow.demand_growth_rate) Cars/Year",
        size = (500, 400),
        color = "teal",
        linewidth = 3,
    )
end
```



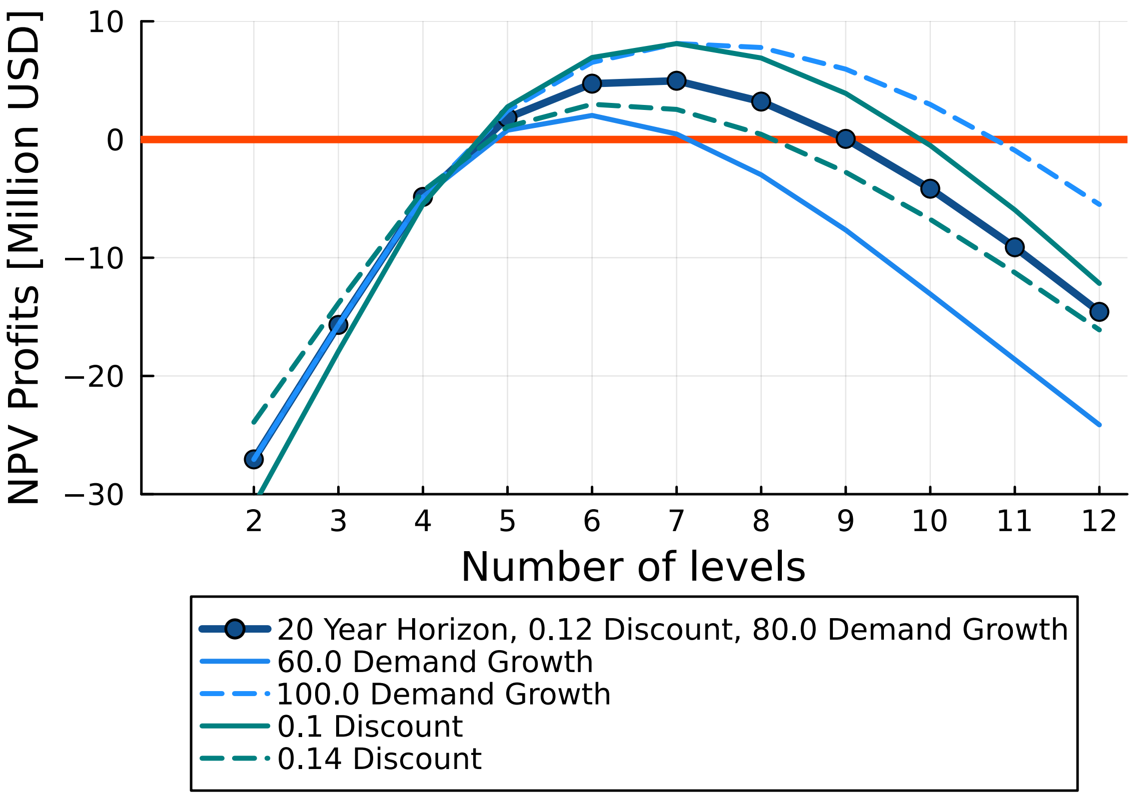
With the demand growth model, a simulation can be performed using a time horizon of 20 years and a discount rate of 12%. Those are considered as constant variables through the analysis considering that they belong to the financial domain exclusively and are probably standards in the "investment" community or they would want to see the simulation results based on those parameters to make decisions. Nevertheless, for visualization purposes two discount rates are simulated to understand the implications.

The simulation uses a *static policy* that add 0 new levels for every time step regardless of the demand quantities. The following figure shows the Net Present Value (NVP) for every design structure level. For preliminary sensitive analysis, growth rates of 60 and 100 are also simulated.

```
let
    level_Δ_a = 0
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.12)
    n_levels = 2:12
    policies = [StaticPolicy(i) for i in n_levels]
    profits = [simulate(sow, level_Δ_a, policy) for policy in policies]
    pl = plot()
    hline!([0]; color = "orangered", linewidth = 3, label = nothing)
    plot!(
```

```
        n_levels,
        profits;
        ylabel = "NPV Profits [Million USD]",
        ylims = [-30,10],
        xlabel = "Number of levels",
        legend = :outerbottom,
        label = "$(sow.n_years) Year Horizon, $(sow.discount_rate) Discount, $(sow.demand_grow
        size = (500, 400),
        marker = :circle,
        xticks = n_levels,
        color = "dodgerblue4",
        linewidth = 3,
    )
    sow = ParkingGarageSOW(; demand_growth_rate=60.0, n_years=20, discount_rate=0.12)
    policies = [StaticPolicy(i) for i in n_levels]
    profits = [simulate(sow, level_Δ_a, policy) for policy in policies]
    plot!(
        n_levels,
        profits;
        label = "$(sow.demand_growth_rate) Demand Growth",
        color = "dodgerblue2",
        linewidth = 2,
    )
    sow = ParkingGarageSOW(; demand_growth_rate=100.0, n_years=20, discount_rate=0.12)
    policies = [StaticPolicy(i) for i in n_levels]
    profits = [simulate(sow, level_Δ_a, policy) for policy in policies]
    plot!(
        n_levels,
        profits;
        label = "$(sow.demand_growth_rate) Demand Growth",
        style = :dash,
        color = "dodgerblue",
        linewidth = 2,
    )
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.10)
    policies = [StaticPolicy(i) for i in n_levels]
    profits = [simulate(sow, level_Δ_a, policy) for policy in policies]
    plot!(
        n_levels,
        profits;
        label = "$(sow.discount_rate) Discount",
        color = "teal",
        linewidth = 2,
    )
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.14)
    policies = [StaticPolicy(i) for i in n_levels]
    profits = [simulate(sow, level_Δ_a, policy) for policy in policies]
```

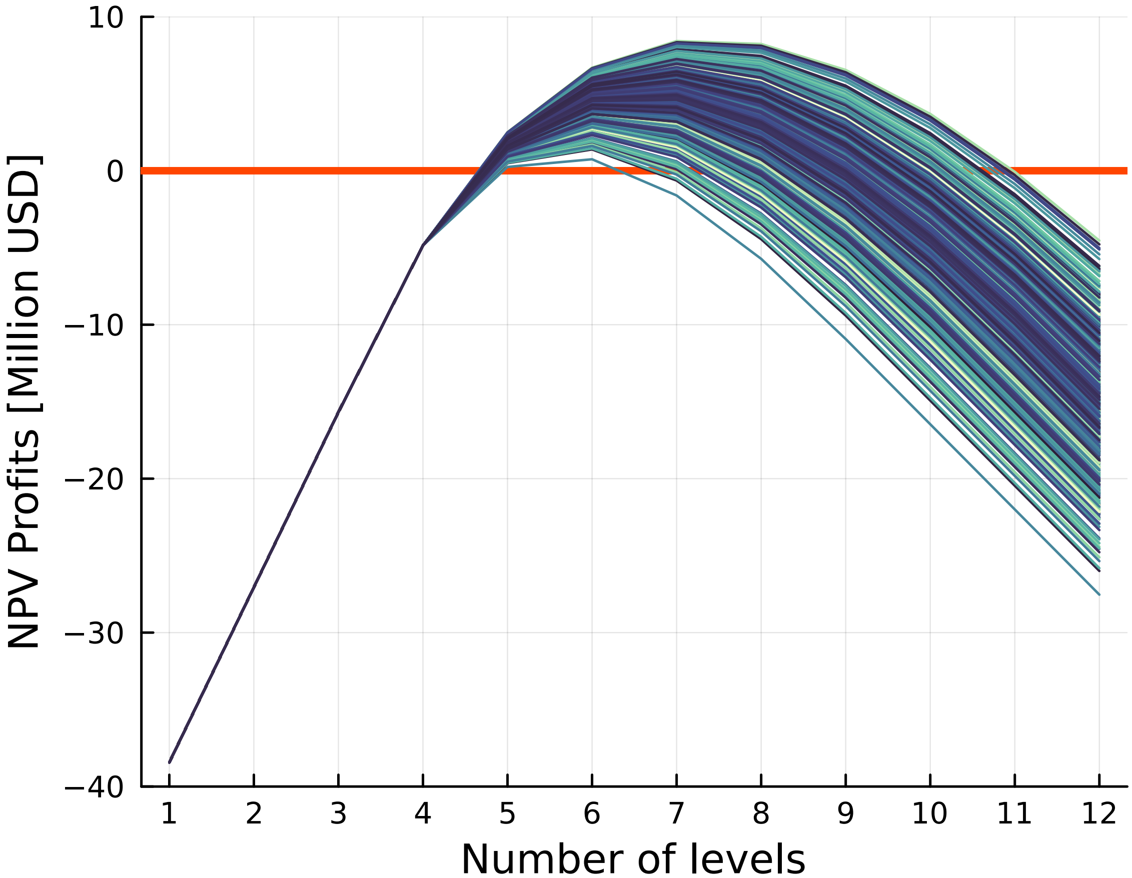
```
    plot!(
        n_levels,
        profits;
        label = "$(sow.discount_rate) Discount",
        color = "teal",
        style = :dash,
        linewidth = 2,
    )
end
```



2 Uncertainty consideration

The uncertainty is to be propagated for the demand growth model. The annual rate is to be sampled from a *Normal* distribution with of 80 and a standard deviation as a parameter of the coefficient of variation (COV). This way, the level of uncertainty can be propagated in the model. As mentioned before, the discount rate and the time horizon are kept constant (12% and 20 years, respectively). The following is a simulation for a COV = 10%.

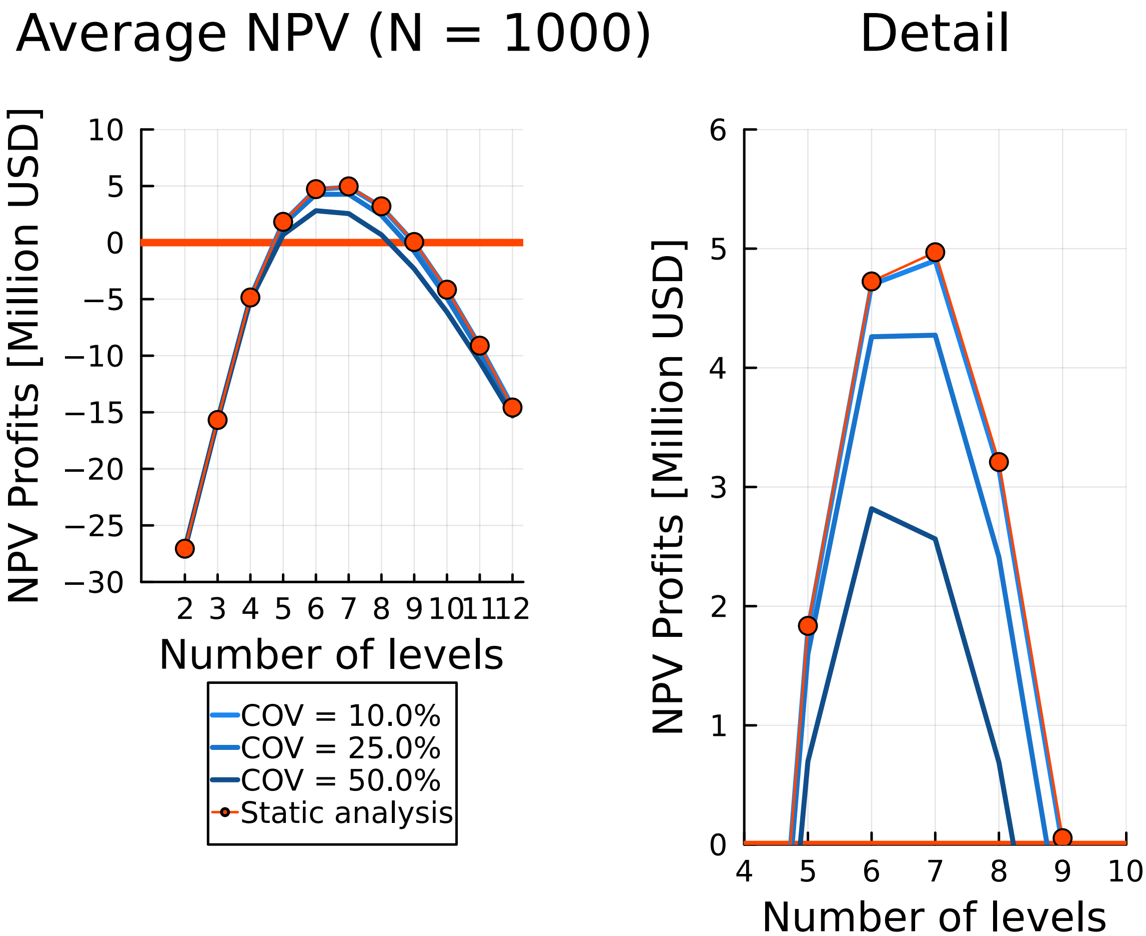
```julia
function draw_growth_rate(COV)
      = 80.0
    return rand(Normal( , COV *  ))
end
let
    level_Δ_a = 0
    COV = 0.10
    n_levels = 1:12
    N_samples = 1000

    pl = plot(;
            ylabel = "NPV Profits [Million USD]",
            ylims = [-40,10],
            xlabel = "Number of levels",
            legend = false,
            size = (500, 400),
            xticks = n_levels,
            linewidth = 1,
            alpha = 0.2,)
    hline!([0]; color = "orangered", linewidth = 3, label = nothing)
    for s in 1:N_samples
        sow = ParkingGarageSOW(; demand_growth_rate = draw_growth_rate(COV),
                                n_years = 20,
                                discount_rate = 0.12)
        policies = [StaticPolicy(i) for i in n_levels]
        profits = [simulate(sow, level_Δ_a, policy) for policy in policies]

        plot!(n_levels,
              profits;
              palette = :deep
             )
    end
    pl
end
```



In order to compare to the deterministic/static case, three different levels of uncertainty (i.e. COV = 10, 25, and 50%) are used. This way, it is clear that the greater the uncertainty of the demand model is the lower expected returns are simulated. The following figure represent the average NPV for a 1000 sampling simulation.

```
let
    level_Δ_a = 0
    covs = [0.10, 0.25, 0.50]
    N_samples = 1000
    n_levels = 2:12

    pl = plot(;
            ylabel = "NPV Profits [Million USD]",
            y_ticks = -30:5:10,
            ylims = [-30,10],
            xlabel = "Number of levels",
            legend = :outerbottom,
            size = (500, 400),
            xticks = n_levels,
            linewidth = 3,
```

```
            title = "Average NPV (N = $N_samples)")
    hline!([0]; color = "orangered", linewidth = 3, label = nothing)
    for s in 1:length(covs)
        profits = zeros(length(n_levels),1)
        for n in 1:N_samples
            sow = ParkingGarageSOW(; demand_growth_rate = draw_growth_rate(covs[s]),
                                    n_years = 20,
                                    discount_rate = 0.12)
            policies = [StaticPolicy(i) for i in n_levels]
            profits .+= [simulate(sow,level_Δ_a, policy) for policy in policies]
        end
        plot!(n_levels,
              profits/N_samples;
              label = "COV = $(100*covs[s])%",
              color = "dodgerblue$(s+1)",
              linewidth = 2,
                )
    end
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.12)
        policies = [StaticPolicy(i) for i in n_levels]
        profits = [simulate(sow,level_Δ_a, policy) for policy in policies]
        plot!(
            n_levels,
            profits;
            label = "Static analysis",
            marker = :circle,
            color = "orangered",
        )
    pl_det = plot(pl;
                  ylims = [0,6],
                  y_ticks = 0:1:6,
                  xlims = [4,10],
                  title = "Detail",
                  legend = false)
    plot(pl, pl_det, layout = 2)
end
```



3 Sequential decisions (adaptive policy/flexible/options)

The following function was written in the `sim` file of the `ParkingGarage` package to set the sequential policy:

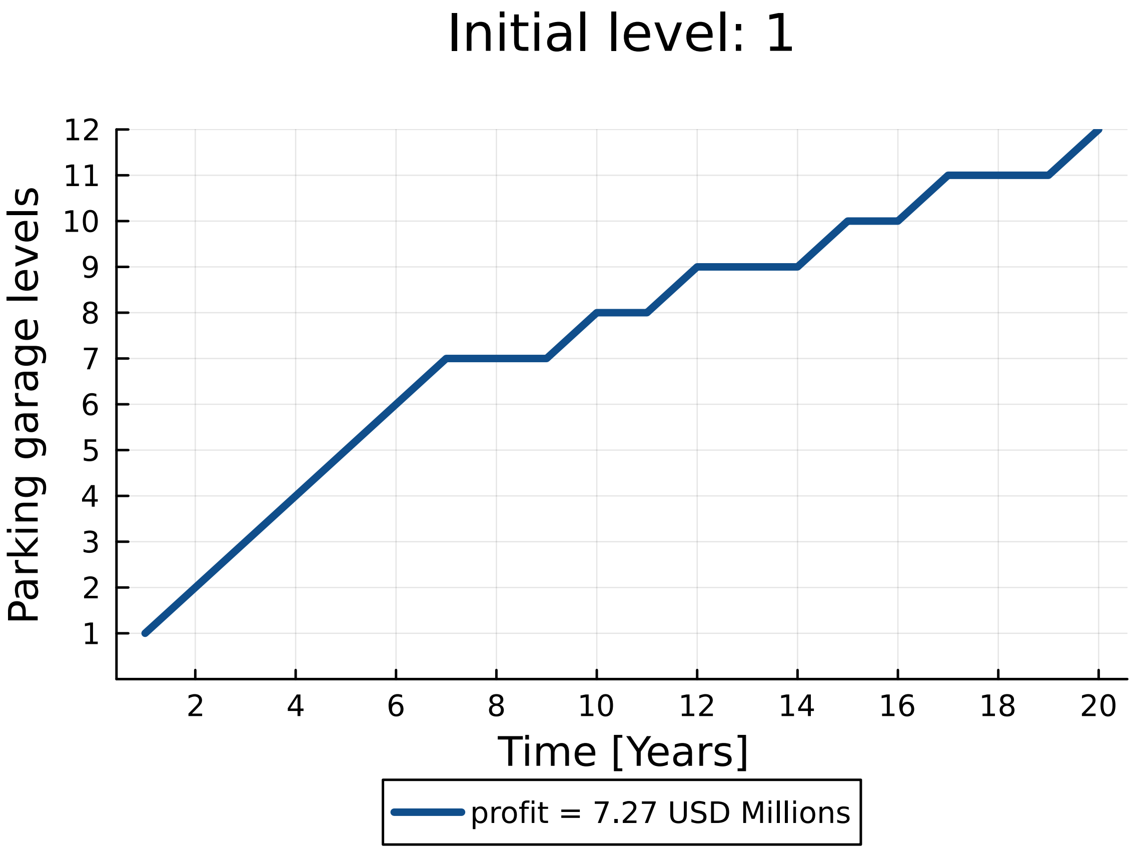
```
let
   function get_action(x::ParkingGarageState, policy::AdaptivePolicy, Δ)
       if x.year == 1
           return ParkingGarageAction(policy.n_levels_init)
       else
           if calculate_capacity(x) < x.demand
               return ParkingGarageAction(Δ)
           else
               return ParkingGarageAction(0)
           end
       end
   end
end
```

The policy consists in building a new level *every time* that the demand surpass the capacity of the parking building.

In order to understand how the sequential decisions are being selected, a new function `simulatelevels` is written in the `sim` file of the `ParkingGarage` package that returns the levels of the parking garage at every time step.

For example, the following are the level changes when the first level decision is one.

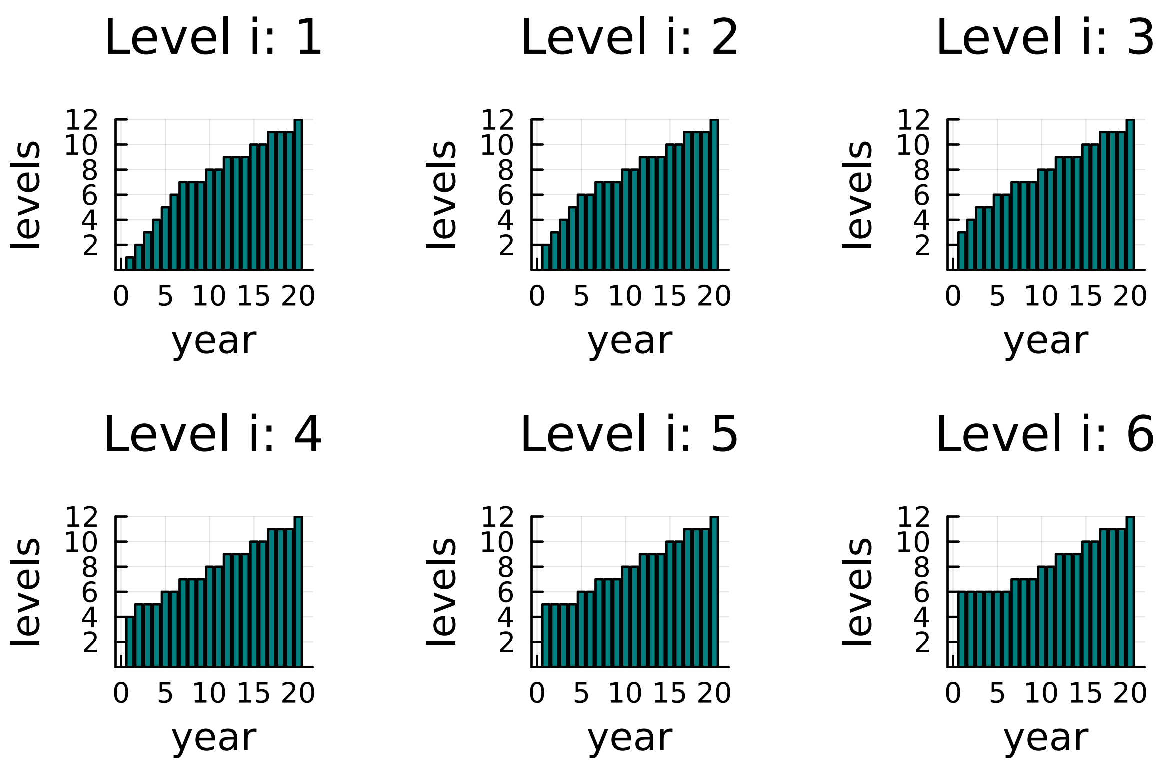
```
let
    years = 20
    level_Δ_a = 1
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=years, discount_rate=0.12)
    n_levels = 1
    policies = [AdaptivePolicy(i) for i in n_levels]
    level_policy = [simulatelevels(sow, level_Δ_a, policy) for policy in policies]
    profits = [simulate(sow, level_Δ_a, policy) for policy in policies]
    pl = plot(1:years,level_policy[1];
              title = "Initial level: $n_levels",
              xlabel = "Time [Years]",
              ylabel = "Parking garage levels",
              label = "profit = $(trunc(profits[1], digits = 2)) USD Millions",
              legend = :outerbottom,
              size = (500, 400),
              ylims = [0,12],
              y_ticks = 1:1:12,
              x_ticks = 2:2:20,
              linewidth = 3,
              color = "dodgerblue4",
              )
    pl
end
```



The following are the sequential decisions for different initial levels (Level i)

```
let
    years = 20
    level_Δ_a = 1
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=years, discount_rate=0.12)
    n_levels = 1:6
    policies = [AdaptivePolicy(i) for i in n_levels]
    level_policy = [simulatelevels(sow, level_Δ_a, policy) for policy in policies]
    profits = [simulate(sow, level_Δ_a, policy) for policy in policies]
    l = @layout [grid(2,3)]
    plot(level_policy;
         layout = l,
         seriestype = [:bar],
         label = nothing,
         color = "teal",
         y_ticks = [2,4,6,8,10,12],
         ylims = [0,12],
         title = ["Level i: $i" for j in 1:1, i in 1:12],
         xlabel = "year",
```

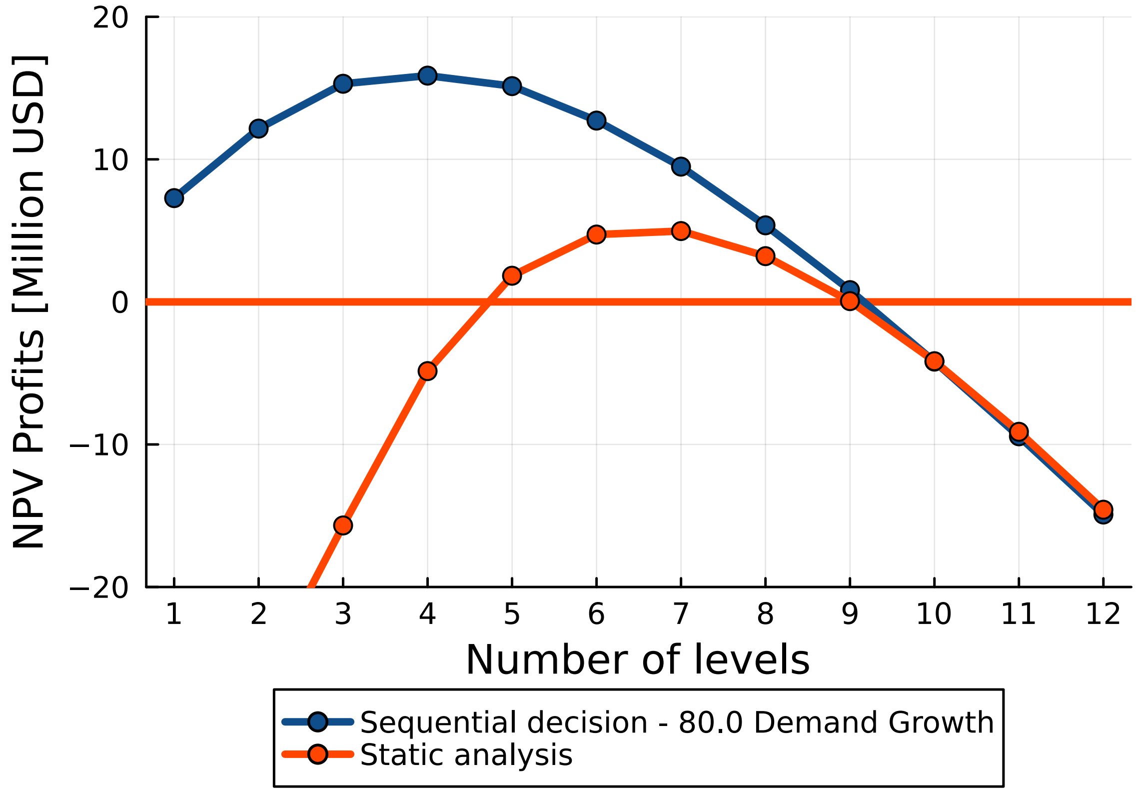
```
        ylabel = "levels",)
end
```



The NPV behavior for this sequential decision proofs to yield to higher profits. It is debatable weather or not it is possible to build a new level every year. If that is the case, for a 80 growth rate demand (static), building 4 levels initially and increasing the number of levels up to 12 (according to the previous figure) yields to >10 USD millions NPV. For that case, starting with one level and keeping growing (~one level every 2 years) is profitable.

```
let
    level_Δ_a = 1
    years = 20
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=years, discount_rate=0.12)
    n_levels = 1:12
    policies = [AdaptivePolicy(i) for i in n_levels]
    profits = [simulate(sow, level_Δ_a, policy) for policy in policies]
    pl = plot()
    hline!([0]; color = "orangered", linewidth = 3, label = nothing)
    plot!(
        n_levels,
        profits;
        ylabel = "NPV Profits [Million USD]",
        ylims = [-20,20],
        xlabel = "Number of levels",
        legend = :outerbottom,
```

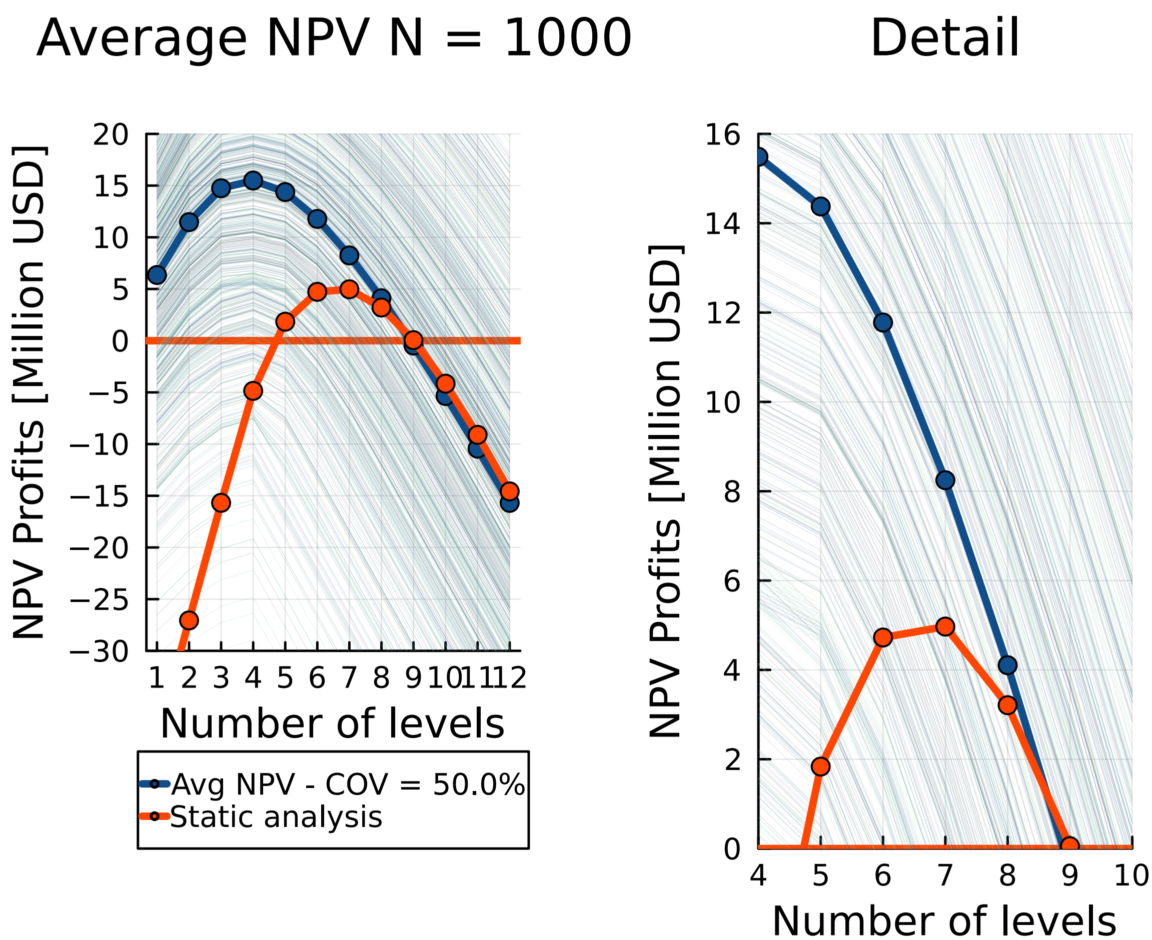
```
        label = "Sequential decision - $(sow.demand_growth_rate) Demand Growth",
        size = (500, 400),
        marker = :circle,
        xticks = n_levels,
        color = "dodgerblue4",
        linewidth = 3,
    )
    level_Δ_a = 0
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.12)
        policies = [StaticPolicy(i) for i in n_levels]
        profits = [simulate(sow,level_Δ_a, policy) for policy in policies]
        plot!(
            n_levels,
            profits;
            label = "Static analysis",
            marker = :circle,
            linewidth = 3,
            color = "orangered",

        )
end
```



When propagating uncertainty in the demand growth, the average NPV shows a similar trend. The following are 1000 simulations for a 50% COV (high uncertainty) and its average NPV. In the background are shown probable results from the simulations.

```
let
    level_Δ_a = 1
    covs = 0.5
    N_samples = 1000
    n_levels = 1:12

    pl = plot(;
            ylabel = "NPV Profits [Million USD]",
            y_ticks = -30:5:20,
            ylims = [-30,20],
            xlabel = "Number of levels",
            legend = :outerbottom,
            size = (500, 400),
            xticks = n_levels,
            linewidth = 3,
            title = "Average NPV N = $N_samples")
```

```
    hline!([0]; color = "orangered", linewidth = 3, label = nothing)

        profits = zeros(length(n_levels),1)
        for n in 1:N_samples
            sow = ParkingGarageSOW(; demand_growth_rate = draw_growth_rate(covs),
                                    n_years = 20,
                                    discount_rate = 0.12)
            policies = [AdaptivePolicy(i) for i in n_levels]
            profit_n = [simulate(sow, level_Δ_a, policy) for policy in policies]
            plot!(n_levels,
                  profit_n;
                  label = nothing,
                  palette = :deep,
                  linewidth = 0.2,
                  alpha = 0.2
                )
            profits .+= profit_n
        end
        plot!(n_levels,
              profits/N_samples;
              label = "Avg NPV - COV = $(100*covs)%",
              color = "dodgerblue4",
              marker = :circle,
              linewidth = 3,
                )
    level_Δ_a = 0
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.12)
        policies = [StaticPolicy(i) for i in n_levels]
        profits = [simulate(sow,level_Δ_a, policy) for policy in policies]
        plot!(
            n_levels,
            profits;
            label = "Static analysis",
            marker = :circle,
            linewidth = 3,
            color = "orangered",

        )
    pl_det = plot(pl;
                  y_ticks = 0:2:16,
                  ylims = [0,16],
                  xlims = [4,10],
                  title = "Detail",
                  legend =false)
    plot(pl, pl_det, layout = 2)
end
```

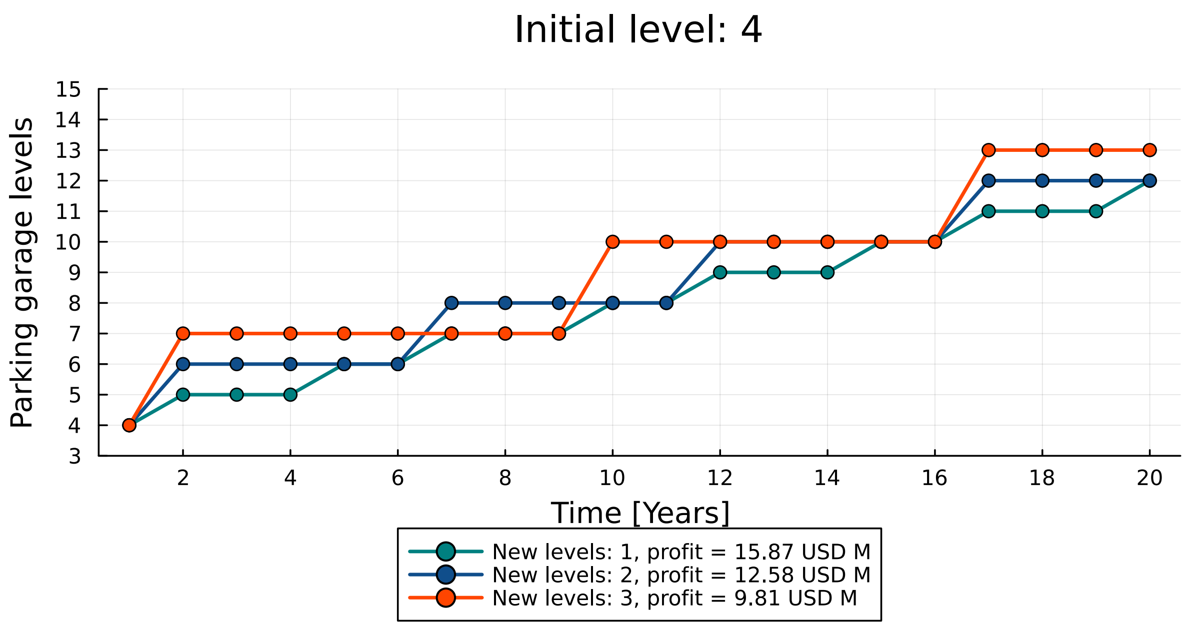


3.1 Number of levels per decision

In order to explore some realities, different sequential policies are used. First, by modifying the `sim` file, it is possible to use as an input the number of levels to build per action. It could be more realistic to assume that building new levels comes with several complexities and problems. It is preferable to performed few alterations to the structure in its timeframe. Here are the comparison of choosing to build 1, 2, and 3 levels when the demand surpasses the capacity.

```
let
    years = 20
    n_levels = 4
    level_Δ_a = [1,2,3]
    pl = plot(;
                title = "Initial level: $n_levels",
                xlabel = "Time [Years]",
                ylabel = "Parking garage levels",
                legend = :outerbottom,
                size = (700, 400),
                ylims = [3,15],
                y_ticks = 3:1:15,
```

```
                x_ticks = 2:2:20,
                linewidth = 3,
                )
    color_plot = ["teal", "dodgerblue4", "orangered"]
    for Δ in level_Δ_a
        sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=years, discount_rate=0.12)

        policies = [AdaptivePolicy(i) for i in n_levels]
        level_policy = [simulatelevels(sow, Δ, policy) for policy in policies]
        profits = [simulate(sow, Δ, policy) for policy in policies]
        pl = plot!(pl,1:years,level_policy[1];
                label = "New levels: $Δ, profit = $(trunc(profits[1], digits = 2)) USD M",
                color = color_plot[Δ],
                linewidth = 2,
                marker = :circle,
                )
    end
    pl
end
```



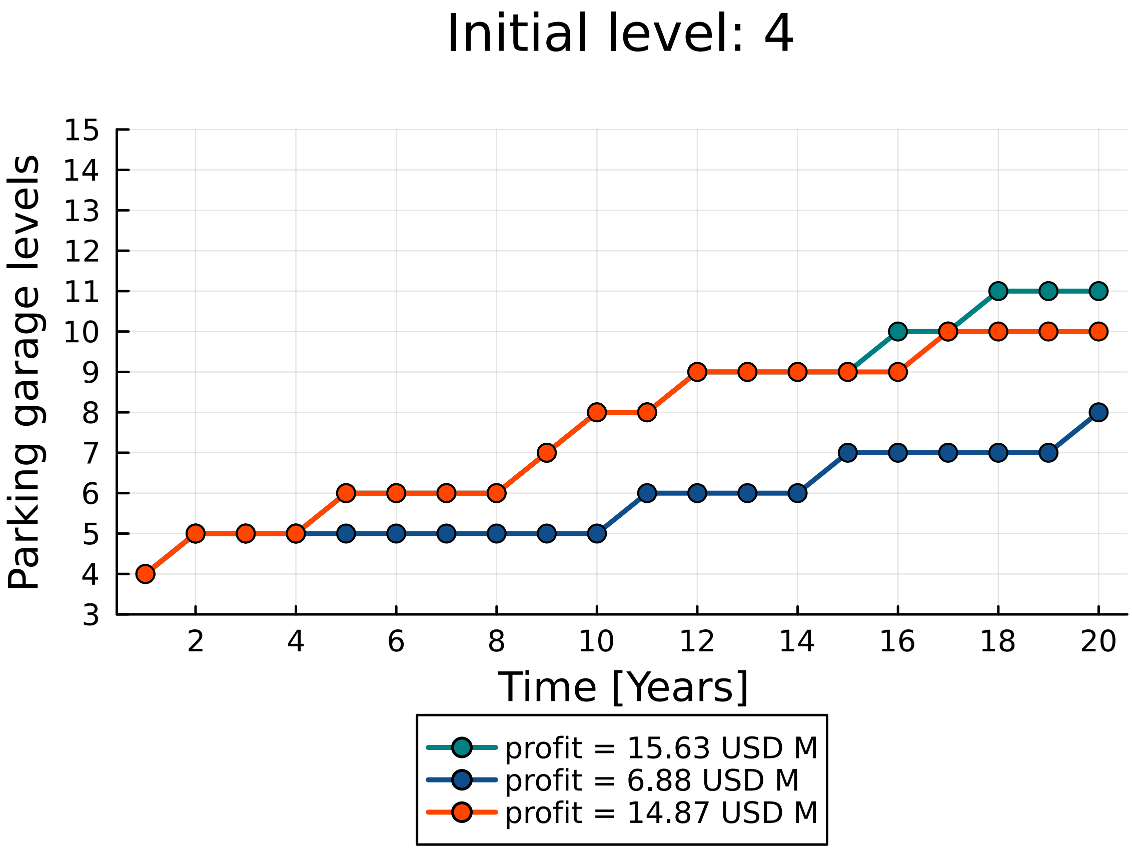
3.2 Random decision +1 level

Another decision policy would be considering that, for different reasons, the owner could or not decide to build a new level *even* if the demand surpass the capacity (in the paper the decision was based on two consecutive years surpassing the capacity). A new `get_action` function is written in `sim` file to build one additional level or not (binary variable) randomly when the demand surpasses the capacity.

```julia
function get_actionbin(x::ParkingGarageState, policy::AdaptivePolicy)
   if x.year == 1
       return ParkingGarageAction(policy.n_levels_init)
   else
       if calculate_capacity(x) < x.demand
           return ParkingGarageAction(trunc(Int, round(rand(1)[1])))
       else
           return ParkingGarageAction(0)
       end
   end
end
```

The following are some sequential decisions for the 80 rate demand growth ratio and four initial levels.

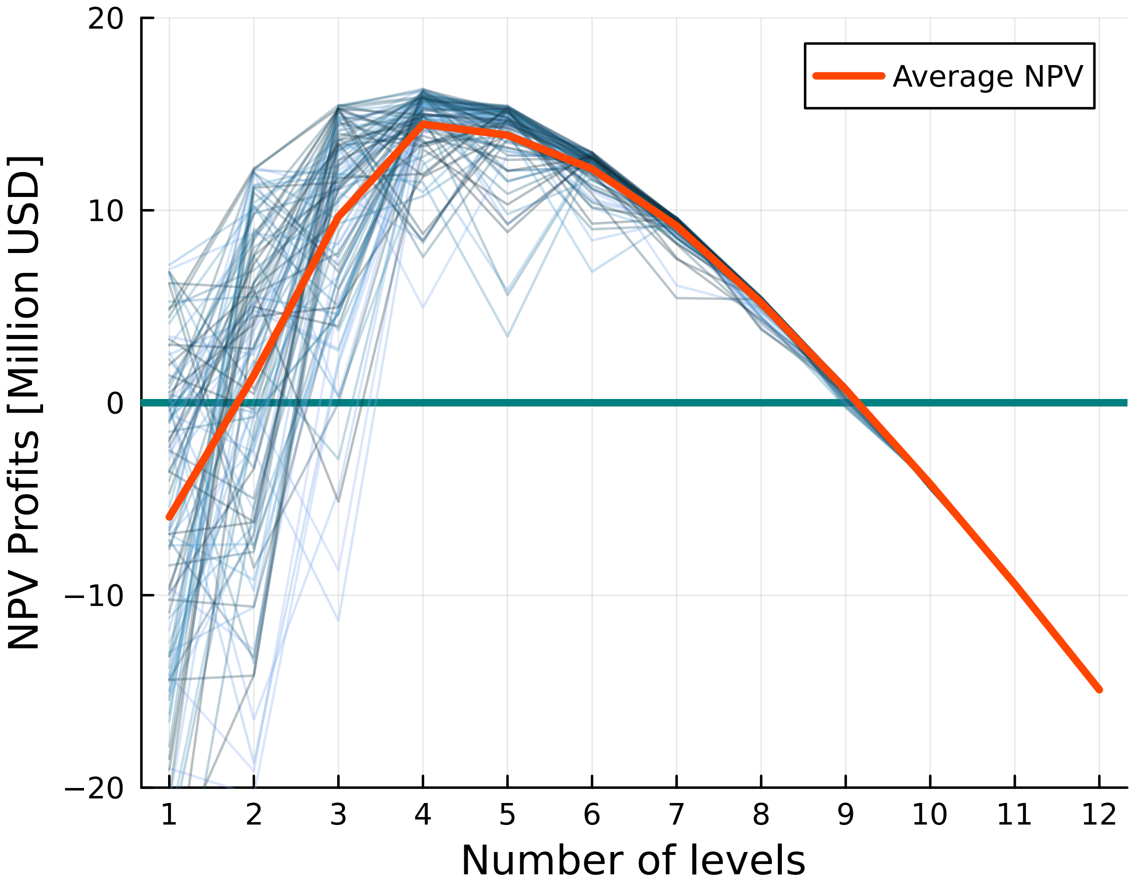
```julia
let
    years = 20
    n_levels = 4
    color_plot = ["teal", "dodgerblue4", "orangered"]
    pl = plot(;
                title = "Initial level: $n_levels",
                xlabel = "Time [Years]",
                ylabel = "Parking garage levels",
                legend = :outerbottom,
                size = (500, 400),
                ylims = [3,15],
                y_ticks = 3:1:15,
                x_ticks = 2:2:20,
                linewidth = 3,
                )
    for i in 1:3
        sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=years, discount_rate=0.12)
        policies = [AdaptivePolicy(i) for i in n_levels]
        level_policy = [simulatelevelsbin(sow, policy) for policy in policies]
        profits = [simulatebin(sow, policy) for policy in policies]
        pl = plot!(pl,1:years,level_policy[1];
                label = "profit = $(trunc(profits[1], digits = 2)) USD M",
                linewidth = 2,
                marker = :circle,
                color = color_plot[i]
                )
    end
    pl
end
```



This way, uncertainty in the decision is added to the model. The following are 100 simulations where there is large uncertainty even for static growth rate model.

```
let
    N_samples = 100
    years = 20
    n_levels = 1:12
    profit_avg = zeros(12,1)
    pl = plot(;
            ylabel = "NPV Profits [Million USD]",
            ylims = [-20,20],
            xlabel = "Number of levels",
            size = (500, 400),
            xticks = n_levels,
            palette = :berlin,
            linewidth = 1,)
    hline!([0]; color = "teal", linewidth = 3, label = nothing)
    for i in 1:N_samples
        sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=years, discount_rate=0.12)
        policies = [AdaptivePolicy(i) for i in n_levels]
```

```
        profits = [simulatebin(sow, policy) for policy in policies]
        plot!(pl,
            n_levels,
            profits,
            label = nothing,
            alpha = 0.3
        )
        profit_avg = profit_avg + profits
    end
    plot!(pl,
          n_levels,
          profit_avg ./ N_samples;
          linewidth = 3,
          color = "orangered",
          label = "Average NPV")
    pl
end
```



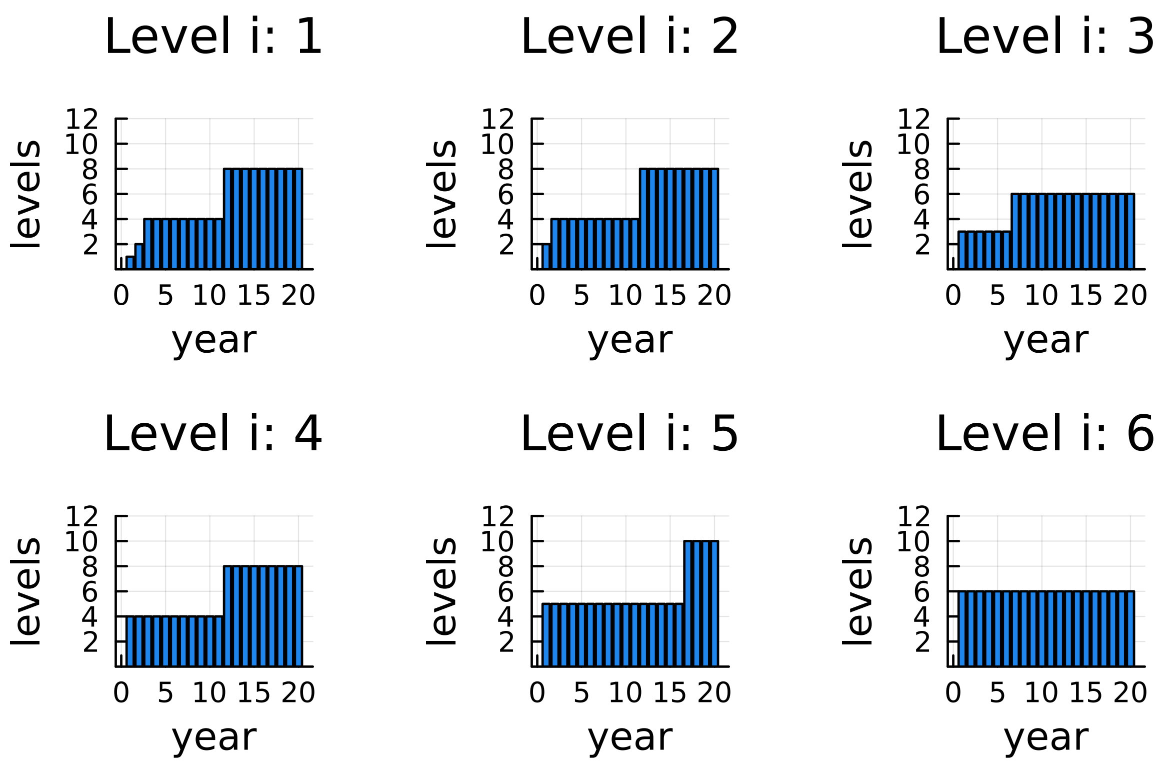
3.3 Minimizing new construction

Another policy could be minimizing new construction but also profiting from the new demand. A new `get_action` function is written to doubles the capacity of the building when the demand doubles the current capacity.

```
function get_action_double(x::ParkingGarageState, policy::AdaptivePolicy)
   if x.year == 1
       return ParkingGarageAction(policy.n_levels_init)
   else
       if calculate_capacity(x) < 0.50 * x.demand
           return ParkingGarageAction(x.n_levels)
       else
           return ParkingGarageAction(0)
       end
   end
end
```

This way, the number of times the garage is in construction is very limited. The following are the sequential decisions under this policy.

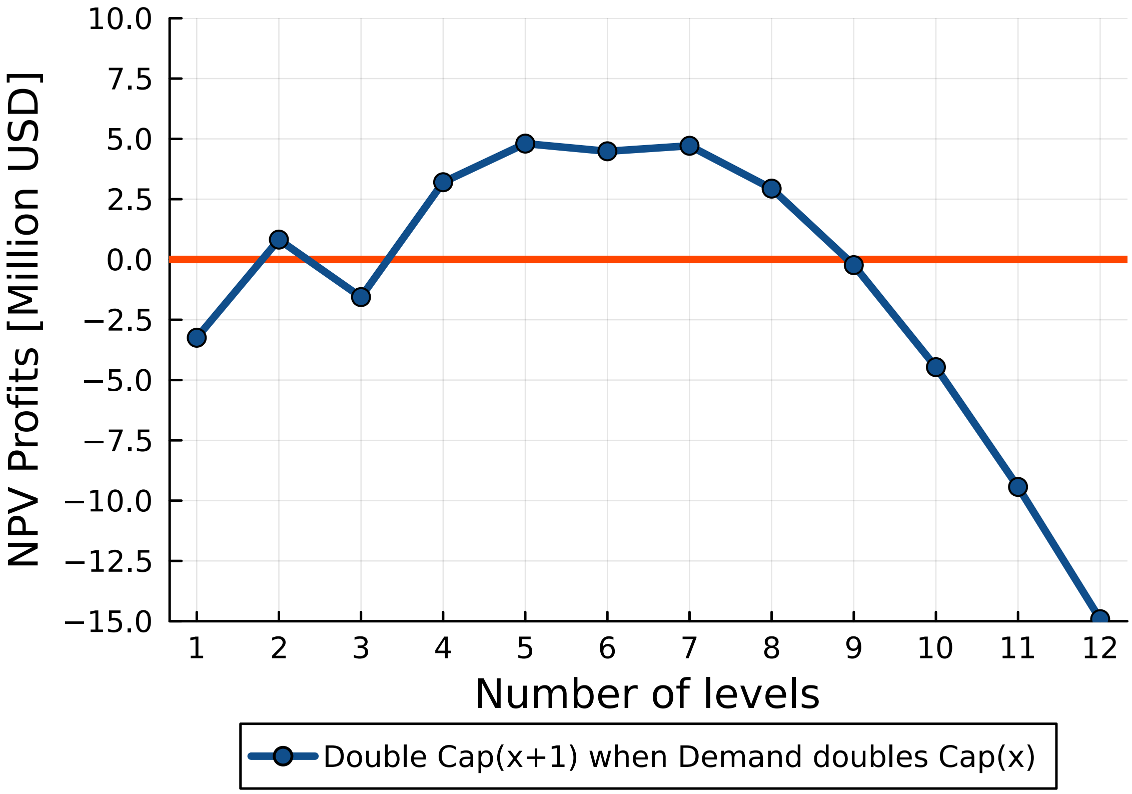
```
let
    years = 20
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=years, discount_rate=0.12)
    n_levels = 1:6
    policies = [AdaptivePolicy(i) for i in n_levels]
    level_policy = [simulatelevels_double(sow, policy) for policy in policies]
    profits = [simulate_double(sow, policy) for policy in policies]
    l = @layout [grid(2,3)]
    plot(level_policy;
         layout = l,
         seriestype = [:bar],
         label = nothing,
         color = "dodgerblue2",
         y_ticks = 2:2:12,
         ylims = [0,12],
         title = ["Level i: $i" for j in 1:1, i in 1:12],
         xlabel = "year",
         ylabel = "levels",)
end
```



The following would be the NPV behavior for a static 80 growth rate demand. The curve shows a flat behavior given that the policy is very similar regardless of the initial level for the 4 to 7 level range.

```
let
    years = 20
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=years, discount_rate=0.12)
    n_levels = 1:12
    policies = [AdaptivePolicy(i) for i in n_levels]
    profits = [simulate_double(sow, policy) for policy in policies]
    pl = plot()
    hline!([0]; color = "orangered", linewidth = 3, label = nothing)
    plot!(
        n_levels,
        profits;
        ylabel = "NPV Profits [Million USD]",
        ylims = [-15,10],
        y_ticks = -15:2.5:10,
        xlabel = "Number of levels",
        legend = :outerbottom,
        label = "Double Cap(x+1) when Demand doubles Cap(x) ",
        size = (500, 400),
        marker = :circle,
        xticks = n_levels,
        color = "dodgerblue4",
```

```
        linewidth = 3,
    )
end
```

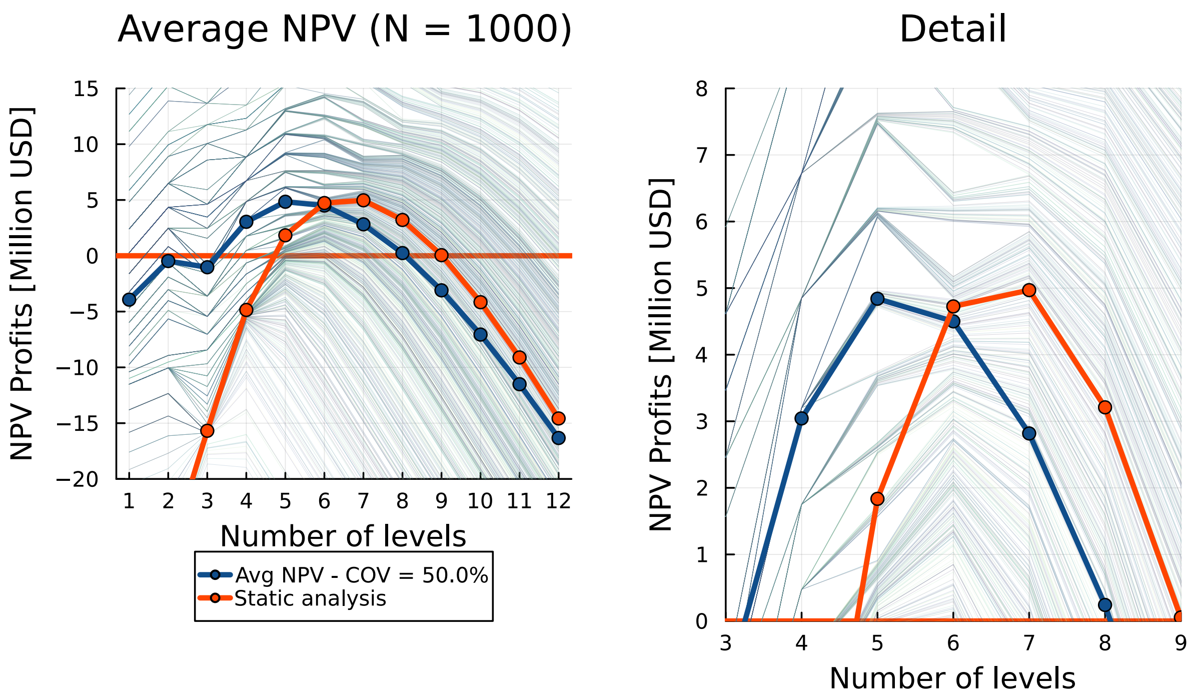


When incorporating the variability from the demand growth model (high uncertainty 50% COV), the following is the average NPV from 1000 simulations.

```
let
    covs = 0.5
    N_samples = 1000
    n_levels = 1:12

    pl = plot(;
            ylabel = "NPV Profits [Million USD]",
            y_ticks = -20:5:15,
            ylims = [-20,15],
            xlabel = "Number of levels",
            legend = :outerbottom,
            size = (700, 400),
            xticks = n_levels,
```

```
            linewidth = 3,
            title = "Average NPV (N = $N_samples)")
    hline!([0]; color = "orangered", linewidth = 3, label = nothing)

        profits = zeros(length(n_levels),1)
        for n in 1:N_samples
            sow = ParkingGarageSOW(; demand_growth_rate = draw_growth_rate(covs),
                                    n_years = 20,
                                    discount_rate = 0.12)
            policies = [AdaptivePolicy(i) for i in n_levels]
            profit_n = [simulate_double(sow, policy) for policy in policies]
            plot!(n_levels,
                  profit_n;
                  label = nothing,
                  palette = :deep,
                  linewidth = 0.2,
                  alpha = 0.2
                )
            profits .+= profit_n
        end
        plot!(n_levels,
              profits/N_samples;
              label = "Avg NPV - COV = $(100*covs)%",
              color = "dodgerblue4",
              marker = :circle,
              linewidth = 3,
                )
    level_Δ_a = 0
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.12)
        policies = [StaticPolicy(i) for i in n_levels]
        profits = [simulate(sow,level_Δ_a, policy) for policy in policies]
        plot!(
            n_levels,
            profits;
            label = "Static analysis",
            marker = :circle,
            linewidth = 3,
            color = "orangered",

        )
    pl_det = plot(pl;
                  y_ticks = 0:1:8,
                  ylims = [0,8],
                  xlims = [3,9],
                  title = "Detail",
                  legend = false)
    plot(pl, pl_det, layout = 2)
```

```
end
```



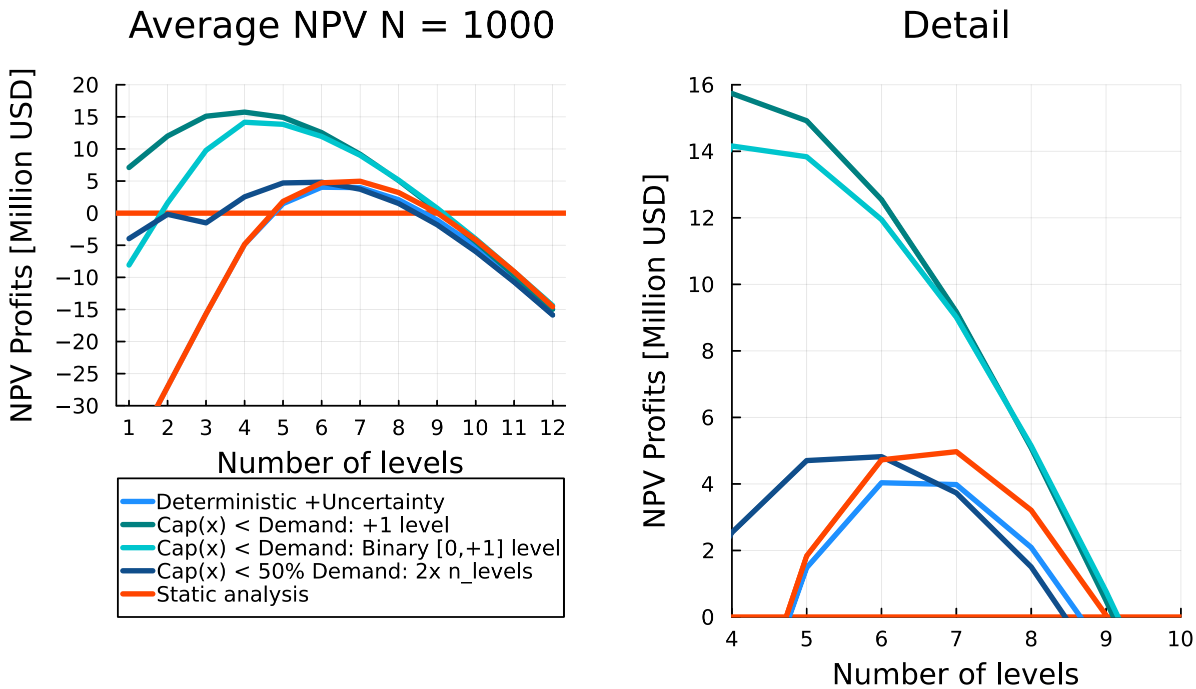
4 Comparisson

When incorporating the variability from the demand growth model (high uncertainty 50% COV), the following is the average NPV from 1000 simulations. Finally, the following is the comparison for the deterministic model (with and without uncertainty), and the studied sequential decision policies. The figure is generated from 1000 samples and a **COV of 30%**.

```
let
    covs = 0.30
    N_samples = 1000
    n_levels = 1:12

    pl = plot(;
            ylabel = "NPV Profits [Million USD]",
            y_ticks = -30:5:20,
            ylims = [-30,20],
            xlabel = "Number of levels",
            legend = :outerbottom,
            size = (700, 400),
            xticks = n_levels,
            linewidth = 3,
            title = "Average NPV N = $N_samples")
    hline!([0]; color = "orangered", linewidth = 3, label = nothing)
        # Uncertainty
        level_Δ_a = 0
```

```julia
        profits = zeros(length(n_levels),1)
        for n in 1:N_samples
            sow = ParkingGarageSOW(; demand_growth_rate = draw_growth_rate(covs),
                                   n_years = 20,
                                   discount_rate = 0.12)
            policies = [StaticPolicy(i) for i in n_levels]
            profits .+= [simulate(sow,level_Δ_a, policy) for policy in policies]
        end
        plot!(n_levels,
              profits/N_samples;
              label = "Deterministic +Uncertainty",
              linewidth = 3,
              color = "dodgerblue1",
                )
        # Building a new level when capacity is surpass by demand
        level_Δ_a = 1
        profits = zeros(length(n_levels),1)
        for n in 1:N_samples
            sow = ParkingGarageSOW(; demand_growth_rate = draw_growth_rate(covs),
                                   n_years = 20,
                                   discount_rate = 0.12)
            policies = [AdaptivePolicy(i) for i in n_levels]
            profit_n = [simulate(sow, level_Δ_a, policy) for policy in policies]
            profits .+= profit_n
        end
        plot!(n_levels,
              profits/N_samples;
              label = "Cap(x) < Demand: +1 level",
              color = "teal",
              linewidth = 3,
                )
        # Binary decision
        profits = zeros(length(n_levels),1)
        for i in 1:N_samples
            sow = ParkingGarageSOW(; demand_growth_rate = draw_growth_rate(covs), n_years = 20
            policies = [AdaptivePolicy(i) for i in n_levels]
            profit_n = [simulatebin(sow, policy) for policy in policies]
            profits .+= profit_n
        end
        plot!(n_levels,
            profits/N_samples;
            linewidth = 3,
            color = "turquoise3",
            label = "Cap(x) < Demand: Binary [0,+1] level")
        # Building when the demand doubles the capacity. Double the capacity
        profits = zeros(length(n_levels),1)
        for n in 1:N_samples
```

```
            sow = ParkingGarageSOW(; demand_growth_rate = draw_growth_rate(covs),
                                    n_years = 20,
                                    discount_rate = 0.12)
            policies = [AdaptivePolicy(i) for i in n_levels]
            profit_n = [simulate_double(sow, policy) for policy in policies]
            profits .+= profit_n
        end
        plot!(n_levels,
              profits/N_samples;
              label = "Cap(x) < 50% Demand: 2x n_levels",
              color = "dodgerblue4",
              linewidth = 3,
                )
    level_Δ_a = 0
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.12)
        policies = [StaticPolicy(i) for i in n_levels]
        profits = [simulate(sow,level_Δ_a, policy) for policy in policies]
        plot!(
            n_levels,
            profits;
            label = "Static analysis",
            linewidth = 3,
            color = "orangered",

        )
    pl_det = plot(pl;
                  y_ticks = 0:2:16,
                  ylims = [0,16],
                  xlims = [4,10],
                  title = "Detail",
                  legend = false)
    plot(pl, pl_det, layout = 2)
end
```



The following are some conclusions:

1. regardless of the adaptive policy, the parking garage can have higher NPV when "options" are incorporated. In particular, they all shown large profit even when starting the building with small number of levels (low initial investment).
2. If consecutive construction in the building is feasible, the profits can grow importantly when the capacity is closely adapting to the demand.
3. When multiple constructions are not desired, smaller NPV are possible but much higher for small number of initial levels. Considering options clearly represent a better investment in comparison to a single decision upfront.