Lab 7: Parking Garage Case Study

Leanh Nguyen

Fri., Mar. 8

Table of contents

# 1. Setup

## 1.1 Load packages

using Revise  
using ParkingGarage  
using Random  
using Distributions

and also regular packages

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

## 1.2 Formal problem framing

The paper uses an exponential growth model for demand, but we’ll use a linear one.

let  
 sow = ParkingGarageSOW()  
 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]",  
 xlabel="Year",  
 legend=false,  
 title="Demand Growth Rate: $(sow.demand\_growth\_rate) Cars/Year",  
 size=(800, 400),  
 marker=:circle,  
 )  
end

# 2. Deterministic Case

## 2.1 Static Policy

This function assumes that the demand is deterministic and that the number of levels is fixed. The decision variable is the number of levels of the garage to build. If we consider a single SOW, we can calculate the NPV of the profits for a given policy.

let  
 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]  
 println(policies)  
 profits = [simulate(sow, policy) for policy in policies]  
 plot(  
 n\_levels,  
 profits;  
 ylabel="NPV Profits [Million USD]",  
 xlabel="Number of levels",  
 legend=false,  
 title="$(sow.n\_years) Year Horizon, $(sow.discount\_rate) Discount, $(sow.demand\_growth\_rate) Demand Growth, Static Policy, Deterministic Case",  
 size=(800, 400),  
 titlefontsize=10,  
 marker=:circle,  
 xticks=n\_levels,  
 )  
 hline!([0])  
end

StaticPolicy[StaticPolicy(2), StaticPolicy(3), StaticPolicy(4), StaticPolicy(5), StaticPolicy(6), StaticPolicy(7), StaticPolicy(8), StaticPolicy(9), StaticPolicy(10), StaticPolicy(11), StaticPolicy(12)]

## 2.2 Adaptive Policy

This function assumes that the demand is deterministic and that the number of levels is adaptive. The decision variable is the number of levels of the garage to build. If we consider a single SOW, we can calculate the NPV of the profits for a given policy.

let  
 sow = ParkingGarageSOW(; demand\_growth\_rate=80.0, n\_years=20, discount\_rate=0.12)  
 n\_levels\_init = 2:12  
 policies = [AdaptivePolicy(i) for i in n\_levels\_init]  
 println(policies)  
 profits = [simulate(sow, policy) for policy in policies]  
 plot(   
 n\_levels\_init,  
 profits;  
 ylabel="NPV Profits [Million USD]",  
 xlabel="Number of levels",  
 legend=false,  
 title="$(sow.n\_years) Year Horizon, $(sow.discount\_rate) Discount, $(sow.demand\_growth\_rate) Demand Growth, Adaptive Policy, Deterministic Case",  
 size=(800, 400),  
 titlefontsize=10,  
 marker=:circle,  
 xticks=n\_levels\_init,  
 )  
 hline!([0])  
end

AdaptivePolicy[AdaptivePolicy(2), AdaptivePolicy(3), AdaptivePolicy(4), AdaptivePolicy(5), AdaptivePolicy(6), AdaptivePolicy(7), AdaptivePolicy(8), AdaptivePolicy(9), AdaptivePolicy(10), AdaptivePolicy(11), AdaptivePolicy(12)]

# 3. Stochastic Case

Figure 1 of (**deneufville\_parkinggarage:2006?**) shows how the NPV changes when uncertainty is added to the model. Reproduce this figure, using our model. Specifically:

1. Generate an ensemble of SOWs. Justify how you are sampling the three parameters (n\_years, demand\_growth\_rate, and discount\_rate). I suggest to keep n\_years as a constant, and perhaps to keep the discount rate constant as well.

# Parameters  
n\_years = 20 # Constant value for the number of years  
discount\_rate = 0.12 # Constant value for the discount rate  
levels = 2:12 # Number of levels range  
  
# Function to generate an ensemble of SOWs  
function generate\_sows(num\_sows::Int)  
 sow\_ensemble = []  
 for \_ in 1:num\_sows  
 # Sample demand growth rate from a normal distribution  
 demand\_growth\_rate = rand(Normal(80.0, 10.0)) # Mean = 80.0, Standard deviation = 10.0  
   
 # Create sow object with sampled parameters  
 sow = ParkingGarageSOW(demand\_growth\_rate=demand\_growth\_rate,  
 n\_years=n\_years,  
 discount\_rate=discount\_rate)  
   
 push!(sow\_ensemble, sow) # update ensemble of SOWs  
 end  
 return sow\_ensemble  
end

generate\_sows (generic function with 1 method)

1. For each SOW, calculate the NPV for each policy.
2. Calculate the average NPV for each number of levels and plot.

Once you’ve implemented this function, you can simulate the adaptive policy and compare the NPV to the static policy. Compare the fixed and adaptive policies for both the deterministic (single SOW) and stochastic (ensemble of SOWs) cases. Plot the NPV as a function of the number of levels for each case. - Deterministic: For all levels, the adaptive policy has a higher average NPV than the static policy - Stochastic: Up to 10 levels, the adaptive policy has a higher average NPV than the static policy. Above 10 levels, both policies have similar NPVs. - Overall, these pieces of data illustrate that implementing adaptive policy for a parking garage is the better choice as seen from its higher NPV. In other words, building a garage that is strong enough to construct more levels in the future (i.e., adaptive policy) is better financially than building a garage with a fixed number of levels that cannot be changed (i.e., static/fixed policy).

# Function to calculate NPV for each policy for a given SOW  
function calculate\_npv(sow, num\_levels)  
 static\_npv = simulate(sow, StaticPolicy(num\_levels))  
 adaptive\_npv = simulate(sow, AdaptivePolicy(num\_levels))  
 return static\_npv, adaptive\_npv  
end  
  
# Function to calculate average NPV for each number of levels across all SOWs  
function calculate\_average\_npv(ensemble, num\_levels)  
 static\_npv\_avg = []  
 adaptive\_npv\_avg = []  
 # Across all SOWs, calculate NPV for each policy  
 for sow in ensemble  
 static\_npv, adaptive\_npv = calculate\_npv(sow, num\_levels)  
 push!(static\_npv\_avg, static\_npv)  
 push!(adaptive\_npv\_avg, adaptive\_npv)  
 end  
 # average NPV  
 static\_npv\_avg = mean(static\_npv\_avg)  
 adaptive\_npv\_avg = mean(adaptive\_npv\_avg)  
 return static\_npv\_avg, adaptive\_npv\_avg  
end  
  
# Number of SOWs -> can be changed  
num\_sows = 10  
  
# Generate an ensemble of SOWs  
ensemble = generate\_sows(num\_sows)  
  
# Define the range of number of levels  
num\_levels\_range = 2:12  
  
# Calculate average NPV for each number of levels  
static\_npv\_avg\_list = []  
adaptive\_npv\_avg\_list = []  
  
for num\_levels in num\_levels\_range  
 static\_npv\_avg, adaptive\_npv\_avg = calculate\_average\_npv(ensemble, num\_levels)  
 push!(static\_npv\_avg\_list, static\_npv\_avg)  
 push!(adaptive\_npv\_avg\_list, adaptive\_npv\_avg)  
end  
  
# Plot the average NPV as a function of the number of levels for both policies  
plot(  
 num\_levels\_range, static\_npv\_avg\_list, label="Static Policy",  
 xlabel="Number of Levels", ylabel="Average NPV",  
 title="Stochastic Cases: Average NPV vs. Number of Levels",  
 titlefontsize=10,  
 marker=:circle,  
)  
plot!(  
 num\_levels\_range, adaptive\_npv\_avg\_list, label="Adaptive Policy",  
 marker=:square,  
)