Lab 7: Parking Garage Case Study

Your Name

Fri., Mar. 8

Table of contents

using Revise  
using ParkingGarage  
using Random  
using Distributions

and also regular packages

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

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

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]  
 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",  
 size=(800, 400),  
 marker=:circle,  
 xticks=n\_levels,  
 )  
 hline!([0])  
end

Generate states of the world:

#Keeping this constant since it's usually constant in buisiness analysis, we   
#want to look at some reasonable time horizon that we hope to get a return by.  
function Gen\_Years()  
 years = 20   
 return years   
end   
  
#Keeping this centered around 0.12 to keep it comparable to the previous graph,  
#But we'll add some uncertainty anyways incase it does something interesting.  
function Gen\_Rates()  
 Discount\_rate = rand(Normal(0.12, 0.01))  
 return Discount\_rate  
end   
  
function Gen\_Growth() #Generate growth rate value   
 Demand\_growth\_rate = rand(Normal(80, 10))  
 return Demand\_growth\_rate  
end

Gen\_Growth (generic function with 1 method)

n\_sow = 50  
  
sows = [  
 ParkingGarageSOW(; demand\_growth\_rate=Gen\_Growth(), n\_years=Gen\_Years(), discount\_rate=Gen\_Rates())  
 for \_ in 1:n\_sow  
]  
  
#println(sows)

50-element Vector{ParkingGarageSOW{Float64, Int64}}:  
 ParkingGarageSOW{Float64, Int64}(79.57189026966826, 20, 0.12403378147790733)  
 ParkingGarageSOW{Float64, Int64}(107.71248732412462, 20, 0.11099989845756623)  
 ParkingGarageSOW{Float64, Int64}(94.71650142278612, 20, 0.12524316945592132)  
 ParkingGarageSOW{Float64, Int64}(71.65499176836569, 20, 0.1345233832798799)  
 ParkingGarageSOW{Float64, Int64}(68.3632102917413, 20, 0.11645943498184201)  
 ParkingGarageSOW{Float64, Int64}(88.56275909758699, 20, 0.11787988409549352)  
 ParkingGarageSOW{Float64, Int64}(90.90699590282978, 20, 0.1285675269986119)  
 ParkingGarageSOW{Float64, Int64}(91.40790491592816, 20, 0.1344824800296514)  
 ParkingGarageSOW{Float64, Int64}(77.82619452891714, 20, 0.13436184612367347)  
 ParkingGarageSOW{Float64, Int64}(82.67160225507564, 20, 0.10100849157516355)  
 ParkingGarageSOW{Float64, Int64}(77.54886728667505, 20, 0.13056606642903076)  
 ParkingGarageSOW{Float64, Int64}(69.58067762442509, 20, 0.12437572642408777)  
 ParkingGarageSOW{Float64, Int64}(91.44692056212321, 20, 0.12739261470281416)  
 ⋮  
 ParkingGarageSOW{Float64, Int64}(88.03833970681775, 20, 0.12865810262654237)  
 ParkingGarageSOW{Float64, Int64}(73.78803585131763, 20, 0.12441207235498192)  
 ParkingGarageSOW{Float64, Int64}(83.40813388875411, 20, 0.10081389789401547)  
 ParkingGarageSOW{Float64, Int64}(97.91279671700133, 20, 0.11596370878318625)  
 ParkingGarageSOW{Float64, Int64}(82.4554238818875, 20, 0.09969297988149545)  
 ParkingGarageSOW{Float64, Int64}(62.21097310033622, 20, 0.13760121339033746)  
 ParkingGarageSOW{Float64, Int64}(71.01920543482848, 20, 0.12090859609593844)  
 ParkingGarageSOW{Float64, Int64}(68.20946895068298, 20, 0.11241485134939265)  
 ParkingGarageSOW{Float64, Int64}(75.02286283278579, 20, 0.117480494427006)  
 ParkingGarageSOW{Float64, Int64}(82.25818512677968, 20, 0.12359853969300535)  
 ParkingGarageSOW{Float64, Int64}(79.01922851225875, 20, 0.1280120742969667)  
 ParkingGarageSOW{Float64, Int64}(102.64988873019706, 20, 0.11697836487048356)

Calculate NPV for each SOW for each policy:

let   
  
 n\_levels = 2:12  
 policies = [StaticPolicy(i) for i in n\_levels]  
  
 profits = []  
 for policy in policies  
 #println(sow)  
 profits\_list = [simulate(sow, policy) for sow in sows]   
 #println(profits\_list)  
 average\_profit = mean(profits\_list) #average profit for that policy given all SOWs  
 push!(profits, average\_profit)  
 end  
  
 plot(  
 n\_levels,  
 profits;  
 ylabel="Average NPV Profits [Million USD]",  
 xlabel="Number of levels",  
 legend=false,  
 title="With Variance, Average Profits for Static Policy",  
 size=(800, 400),  
 marker=:circle,  
 xticks=n\_levels,  
 )  
 hline!([0])  
  
  
end

Now lets look at NPF for adaptive policies, given a static SOW:

let  
 sow = ParkingGarageSOW(; demand\_growth\_rate=80.0, n\_years=20, discount\_rate=0.12)  
 n\_levels = 1:12  
 policies = [AdaptivePolicy(i) for i in n\_levels]  
 profits = [simulate(sow, policy) for policy in policies]  
 plot(  
 n\_levels,  
 profits;  
 ylabel="NPV Profits [Million USD]",  
 xlabel="Number of levels",  
 legend=false,  
 title="Adaptive Policy, $(sow.n\_years) Year Horizon, $(sow.discount\_rate) Discount, $(sow.demand\_growth\_rate) Demand Growth",  
 size=(800, 400),  
 marker=:circle,  
 xticks=n\_levels,  
 )  
 hline!([0])  
end

Now lets look at NPV for adaptive policies, given different SOWs:

let   
  
 n\_levels = 1:12  
 policies = [AdaptivePolicy(i) for i in n\_levels]  
  
 profits = []  
 for policy in policies  
 #println(sow)  
 profits\_list = [simulate(sow, policy) for sow in sows]   
 #println(profits\_list)  
 average\_profit = mean(profits\_list) #average profit for that policy given all SOWs  
 push!(profits, average\_profit)  
 end  
  
 plot(  
 n\_levels,  
 profits;  
 ylabel="Average NPV Profits [Million USD]",  
 xlabel="Number of levels",  
 legend=false,  
 title="With Variance, Average profits for Adaptive Policy",  
 size=(800, 400),  
 marker=:circle,  
 xticks=n\_levels,  
 )  
 hline!([0])  
  
  
end