Lab 5: Sea-Level Rise

Kyle Olcott kto1

Fri., Feb. 16

Table of contents

# 1. Setup

## 1.1 The usual

As always:

1. Clone the lab repository to your computer
2. Open the lab repository in VS Code
3. Open the Julia REPL and activate, then instantiate, the lab environment
4. Make sure you can render: quarto render template.qmd in the terminal.
   * If you run into issues, try running ] build IJulia in the Julia REPL (] enters the package manager).
   * If you still have issues, try opening up blankfile.py. That should trigger VS Code to give you the option to install the Python extension, which you should do. Then you should be able to open a menu in the bottom right of your screen to select which Python installation you want VS Code to use.

## 1.2 Load packages

using CSV  
using DataFrames  
using DataFramesMeta  
using Distributions  
using Plots  
using StatsPlots  
using Unitful  
  
Plots.default(; margin=5Plots.mm)

## 1.3 Local package

using Revise  
using HouseElevation

house = let  
 haz\_fl\_dept = CSV.read("data/haz\_fl\_dept.csv", DataFrame) # read in the file  
 desc = "Apartment, living area on one floor, Structure"  
 row = @rsubset(haz\_fl\_dept, :Description == desc)[1, :] # select the row I want  
 area = 1033u"ft^2"  
 height\_above\_gauge = 4u"ft"  
 House(  
 row;  
 area=area,  
 height\_above\_gauge=height\_above\_gauge,  
 value\_usd=316\_298,  
 )  
end

House{Int64}(1033, 316298, 4, DepthDamageFunction{Interpolations.Extrapolation{Float64, 1, Interpolations.GriddedInterpolation{Float64, 1, Vector{Float64}, Interpolations.Gridded{Interpolations.Linear{Interpolations.Throw{Interpolations.OnGrid}}}, Tuple{Vector{Float64}}}, Interpolations.Gridded{Interpolations.Linear{Interpolations.Throw{Interpolations.OnGrid}}}, Interpolations.Flat{Nothing}}}(29-element extrapolate(interpolate((::Vector{Float64},), ::Vector{Float64}, Gridded(Linear())), Flat()) with element type Float64:  
 0.0  
 0.0  
 0.0  
 0.0  
 5.0  
 18.0  
 25.0  
 30.0  
 34.0  
 38.0  
 41.0  
 43.0  
 46.0  
 ⋮  
 55.0  
 57.0  
 59.0  
 59.0  
 60.0  
 63.0  
 65.0  
 66.0  
 67.0  
 68.0  
 69.0  
 70.0), String7("RES3"), "206", String31("USACE - Galveston"), "Apartment, living area on one floor, Structure", "missing")

## 1.4 Location

This location is near Peir 12 in Galveston of 2016 Strand #8, Galveston, TX 77550, which is a small groundlevel apartment. Values for the square footage and value were sourced from Redfin. The depth damage curve is thus an apartment on one floor in Galveston.

let  
 depths = uconvert.(u"ft", (-7.0u"ft"):(1.0u"inch"):(30.0u"ft"))  
 damages = house.ddf.(depths) ./ 100  
 damages\_1000\_usd = damages .\* house.value\_usd ./ 1000  
 scatter(  
 depths,  
 damages\_1000\_usd;  
 xlabel="Flood Depth",  
 ylabel="Damage (Thousand USD)",  
 label="$(house.description)\n($(house.source))",  
 legend=:bottomright,  
 size=(800, 400),  
 yformatter=:plain, # prevents scientific notation  
 )  
end

let  
 elevations = 0u"ft":0.25u"ft":14u"ft"  
 costs = [elevation\_cost(house, eᵢ) for eᵢ in elevations]  
 scatter(  
 elevations,  
 costs ./ 1\_000;  
 xlabel="Elevation",  
 ylabel="Cost (Thousand USD)",  
 label="$(house.description)\n($(house.source))",  
 legend=:bottomright,  
 size=(800, 400),  
 yformatter=:plain, # prevents scientific notation  
 )  
end

slr\_scenarios = let  
 df = CSV.read("data/slr\_oddo.csv", DataFrame)  
 [Oddo17SLR(a, b, c, tstar, cstar) for (a, b, c, tstar, cstar) in eachrow(df)]  
end  
println("There are $(length(slr\_scenarios)) parameter sets")

There are 34895 parameter sets

let  
 years = 1900:2150  
 p = plot(;  
 xlabel="Year",  
 ylabel="Mean sea-level (ft)\nwith respect to the year 2000",  
 label="Oddo et al. (2017)",  
 legend=false  
 )  
 for s in rand(slr\_scenarios, 250)  
 plot!(p, years, s.(years); color=:lightgrey, alpha=0.5, linewidth=0.5)  
 end  
 p  
end

# 2. Functions that samples storm surge projections and discount rates.

function draw\_surge\_distribution()  
 μ = rand(Normal(5, 1))  
 σ = rand(Exponential(1.5))  
 ξ = rand(Normal(0.1, 0.05))  
 GeneralizedExtremeValue(μ, σ, ξ)  
end

draw\_surge\_distribution (generic function with 1 method)

function draw\_discount\_rate()  
 return rand(Normal(0.1, 0.02))  
end

draw\_discount\_rate (generic function with 1 method)

## 2.1 Modeling Reasoning

The discount rate is modeled after the S&P 500 which has an expected discount rate of 10%. This may flucuate from time to time by approximately 4%, so the standard deviation will be 2%, allowing for the majority of the normal distribution to be with +/- 4% of 10%.

The surge/flood distribution is hypothetical and based on the deterministic distribution that has been used in previous labs.

p = ModelParams(  
 house=house,  
 years=2024:2083  
)

ModelParams(House{Int64}(1033, 316298, 4, DepthDamageFunction{Interpolations.Extrapolation{Float64, 1, Interpolations.GriddedInterpolation{Float64, 1, Vector{Float64}, Interpolations.Gridded{Interpolations.Linear{Interpolations.Throw{Interpolations.OnGrid}}}, Tuple{Vector{Float64}}}, Interpolations.Gridded{Interpolations.Linear{Interpolations.Throw{Interpolations.OnGrid}}}, Interpolations.Flat{Nothing}}}(29-element extrapolate(interpolate((::Vector{Float64},), ::Vector{Float64}, Gridded(Linear())), Flat()) with element type Float64:  
 0.0  
 0.0  
 0.0  
 0.0  
 5.0  
 18.0  
 25.0  
 30.0  
 34.0  
 38.0  
 41.0  
 43.0  
 46.0  
 ⋮  
 55.0  
 57.0  
 59.0  
 59.0  
 60.0  
 63.0  
 65.0  
 66.0  
 67.0  
 68.0  
 69.0  
 70.0), String7("RES3"), "206", String31("USACE - Galveston"), "Apartment, living area on one floor, Structure", "missing"), [2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033 … 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083], 10000)

sow = SOW(  
 rand(slr\_scenarios),  
 draw\_surge\_distribution(),  
 draw\_discount\_rate()  
)

SOW{Float64}(Oddo17SLR{Float64}(19.12456757, 1.653829655, -0.00131388, 2078.736936, 28.19253359), GeneralizedExtremeValue{Float64}(μ=5.763201212952459, σ=6.765536425469831, ξ=0.10300345472900643), 0.13284114916868037)

a = Action(5.0u"ft")

Action{Float64}(5.0)

res = run\_sim(a, sow, p)

-580779.8293950995

tempsows = [SOW(rand(slr\_scenarios), draw\_surge\_distribution(), draw\_discount\_rate()) for \_ in 1:10] # for 10 SOWs  
sows = []  
for i in 1:11  
append!(sows, tempsows)  
end  
  
n = 0  
actions = []  
while n < 11  
tempactions = [Action(n\*1u"ft") for \_ in 1:10] # these are all the same  
append!(actions, tempactions)  
n = n+1  
end  
  
results = [run\_sim(a, s, p) for (a, s) in zip(actions, sows)]

110-element Vector{Float64}:  
 -720808.5960653564  
 -323246.3145348386  
 -1.4546975571654555e6  
 -727410.4744601309  
 -975675.4839129061  
 -738650.7532591143  
 -477872.658493527  
 -914693.0014890146  
 -762421.7474843427  
 -738982.5388511322  
 -532004.3681767707  
 -195666.10855700303  
 -1.236901324386961e6  
 ⋮  
 -252708.53575086128  
 -112424.18926472458  
 -117731.56772767496  
 -117612.23105583258  
 -117588.75  
 -124142.38233608376  
 -117594.9950195335  
 -117588.75  
 -117589.33458276629  
 -117588.8463520417  
 -231425.83775442938  
 -117589.05067137726

df = DataFrame(  
 npv=results,  
 Δh\_ft=[a.Δh\_ft for a in actions],  
 slr\_a=[s.slr.a for s in sows],  
 slr\_b=[s.slr.b for s in sows],  
 slr\_c=[s.slr.c for s in sows],  
 slr\_tstar=[s.slr.tstar for s in sows],  
 slr\_cstar=[s.slr.cstar for s in sows],  
 surge\_μ=[s.surge\_dist.μ for s in sows],  
 surge\_σ=[s.surge\_dist.σ for s in sows],  
 surge\_ξ=[s.surge\_dist.ξ for s in sows],  
 discount\_rate=[s.discount\_rate for s in sows],  
)

let  
 npv = results  
 elevated = [a.Δh\_ft for a in actions]  
 scatter(  
 elevated,  
 npv;  
 xlabel="Elevation Action",  
 ylabel="NPV",  
 label="The NPV for elevating an apartment at different elevations under 10 different SOW's",  
 legend=:bottomright,  
 size=(800, 400),  
 yformatter=:plain, # prevents scientific notation  
 )  
end

let  
 npv = results  
 discountrate = [s.discount\_rate for s in sows]  
 scatter(  
 discountrate,  
 npv;  
 xlabel="Discount Rate",  
 ylabel="NPV",  
 label="The NPV under 10 different SOW's compared to the discount rate (%)",  
 legend=:bottomright,  
 size=(800, 400),  
 yformatter=:plain, # prevents scientific notation  
 )  
end

npv = results  
 surge = DataFrame(surge\_μ=[s.surge\_dist.μ for s in sows],  
 surge\_σ=[s.surge\_dist.σ for s in sows],  
 surge\_ξ=[s.surge\_dist.ξ for s in sows],)  
 damage = []  
 for i in 1:110  
 total = 0  
 for n in 1:10000  
 sample = rand(GeneralizedExtremeValue(surge[i, 1], surge[i, 2], surge[i, 3]))  
 total = total + sample  
 global dmean = total/10000  
 end  
 damage = [damage;dmean]  
 end  
 scatter(  
 damage,  
 npv;  
 xlabel="Expected Damage without elevating or sea rise (ft)",  
 ylabel="NPV",  
 label="The NPV under 10 different SOW's compared to the expected flood height from the gauge",  
 legend=:bottomright,  
 size=(800, 400),  
 yformatter=:plain, # prevents scientific notation  
 )

# 3. Discussion/Analysis

Three scatter plots were created to test to what extent the different parameters influenced the final outcome of the NPV. For the analysis, 10 different SOW’s were created. Under each of these SOW’s 11 actions were performed in each of them by elevating the house from 0 ft to 10 ft.

The most important parameter was the elevation action. On the graph, there is a very clear cone of uncertainty with the NPV rising the more you elevate, which makes sense as less money is spent on rebuilding each year.

The second most important variable was the expected flood height, which approximates the flood distribution in general. For lower heights, the bulk of the points are near the higher NPV values. As the height increase, however, while you do still find points at high NPV values, they are more spread downwards towards lower NPV values. This makes sense as a lower height will result in lower damages in the future.

For the discount rate, there was no meaningful insight that could be gained from the graph as the points were spread out fairly uniformily. I expected higher discount rates to have higher NPV values as costs to rebuild in the future would result in a lower subtraction from the NPV.

I couldn’t think of a satisfying way to analyze sea rise as unlike the other parameters, it is something that changes throughout the SOW. While the different coefficients in the equation could be graphed, I was unsure what to expect due to my unfamiliarity with the equation.

Overall, the analysis shows that it is almost always a good decision to elevate yours house under a range of SOWs. Given unlimited computing power it would make sense to compute many more SOW’s and actions as they could better reveal the relationships between the parameters and the NPV. For example, narrowing down on the cone of uncertainty in the NPV vs. Elevation graph. Due to real world computing limitations, though, time to run the program would be the main limiting factor in actually running more SOW’s.