Lab 5: Sea-Level Rise

<Anna Delesalle (ajd16)>

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

# 2. Exploratory Modeling

## 2.1 Apply the model to your site

1. Build your own house object, based on the house you’ve been using (or you can switch if you’d like)

house = let  
 haz\_fl\_dept = CSV.read("data/haz\_fl\_dept.csv", DataFrame) # read in the file  
 desc = "Cafeteria Restaurant, structure"  
 row = @rsubset(haz\_fl\_dept, :Description == desc)[1, :] # select the row I want  
 area = 672u"ft^2"  
 height\_above\_gauge = 4u"ft"  
 House(  
 row;  
 area = area,  
 height\_above\_gauge = height\_above\_gauge,  
 value\_usd = 500\_000,  
 )  
end

House{Int64}(672, 500000, 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  
 0.0  
 15.0  
 18.0  
 20.0  
 23.0  
 25.0  
 27.0  
 28.0  
 30.0  
 ⋮  
 58.0  
 64.0  
 68.0  
 72.0  
 76.0  
 80.0  
 84.0  
 88.0  
 92.0  
 96.0  
 100.0  
 100.0), String7("COM8"), "504", String31("USACE - Galveston"), "Cafeteria Restaurant, structure", "missing")

a. Briefly explain where you got the area, value, and depth-damage curve from

For the area, I was able to calculate it using the scale on Google Maps (the building is a rectangle so I multiplied the length of both sides and adjusted the values to their real size to get the area). The value was an approximated guess looking at the values of current house listings in the same neighborhood as Katie’s Seafood House on Zillow. As for the depth-damage curve, I use the code provided in the instructions which uses the House object I created above with information specific to the building I am looking at.

b. Plot the depth-damage curve

let  
 depths = uconvert.(u"ft", (-7.0u"ft"):(1.0u"inch"):(30.0u"ft")) # =formating the graph  
 damages = house.ddf.(depths) ./ 100 # "house" =object defined above and "ddf" =defined in a different file  
 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

c. Plot the cost of raising the house to different elevations from 0 to 14 ft

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

1. Read in the sea-level rise data

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)] # =reformating the data set as a table  
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

1. Modify my code to create a function to draw samples of storm surge and the discount rate. Explain your modeling choices!

Creating the function: Here I have changed μ to range between 15ft and 20ft because this is the mean height of storm surges around Galveston according to the National Oceanic and Atmospheric Administration https://www.nhc.noaa.gov/surge/

function draw\_surge\_distribution()   
 μ = rand(Normal(20, 15)) # =location parameter (mean/most likely value)  
 σ = rand(Exponential(1.5)) # =scale parameter  
 ξ = rand(Normal(0.1, 0.05)) # =tail  
 GeneralizedExtremeValue(μ, σ, ξ)  
end

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

[draw\_surge\_distribution() for \_ in 1:1000]

1000-element Vector{GeneralizedExtremeValue{Float64}}:  
 GeneralizedExtremeValue{Float64}(μ=9.153898451723046, σ=1.294572795100315, ξ=-0.0005069258249926073)  
 GeneralizedExtremeValue{Float64}(μ=14.905116170495756, σ=0.641734530712069, ξ=0.0021854885807456315)  
 GeneralizedExtremeValue{Float64}(μ=21.4665358649923, σ=0.9958424373685817, ξ=0.1371900913911433)  
 GeneralizedExtremeValue{Float64}(μ=21.35386452040847, σ=3.088070090478536, ξ=0.14820957984601577)  
 GeneralizedExtremeValue{Float64}(μ=18.222686046405652, σ=3.7341924842825347, ξ=0.13002415569766446)  
 GeneralizedExtremeValue{Float64}(μ=22.61990399835577, σ=0.7074433150542966, ξ=0.17069103947569075)  
 GeneralizedExtremeValue{Float64}(μ=23.197170067979293, σ=1.5783062175627611, ξ=0.09697870676184796)  
 GeneralizedExtremeValue{Float64}(μ=9.29509098356618, σ=3.6093088176942203, ξ=0.13209434473565193)  
 GeneralizedExtremeValue{Float64}(μ=23.606982557690934, σ=0.22872190620291535, ξ=0.054910890184125025)  
 GeneralizedExtremeValue{Float64}(μ=19.852399357088718, σ=0.5790620206033694, ξ=0.11939042795604445)  
 GeneralizedExtremeValue{Float64}(μ=12.639780833509999, σ=4.287613411816249, ξ=0.10596630362932853)  
 GeneralizedExtremeValue{Float64}(μ=35.15092858710179, σ=6.429025554266745, ξ=0.23237614310305624)  
 GeneralizedExtremeValue{Float64}(μ=28.84711817351804, σ=0.2852781844325368, ξ=0.1079717553142324)  
 ⋮  
 GeneralizedExtremeValue{Float64}(μ=34.18263347942865, σ=0.349604569183831, ξ=0.03383088500269109)  
 GeneralizedExtremeValue{Float64}(μ=9.870087452260604, σ=1.6178470714168927, ξ=0.04962934182258655)  
 GeneralizedExtremeValue{Float64}(μ=40.15341716374208, σ=2.18390446343065, ξ=0.05225331581648566)  
 GeneralizedExtremeValue{Float64}(μ=31.289093361065248, σ=2.0899081295112074, ξ=0.11153789506745866)  
 GeneralizedExtremeValue{Float64}(μ=1.6054738222728027, σ=1.3838795515333708, ξ=0.10151625753557159)  
 GeneralizedExtremeValue{Float64}(μ=14.467484745684956, σ=0.5898018462406425, ξ=0.13999852526177206)  
 GeneralizedExtremeValue{Float64}(μ=3.8258720953931586, σ=0.34457265105484236, ξ=0.1041780667117051)  
 GeneralizedExtremeValue{Float64}(μ=37.104438857960744, σ=2.100081610474562, ξ=0.11365603279975617)  
 GeneralizedExtremeValue{Float64}(μ=30.11830615808804, σ=2.213069738642673, ξ=0.1802543666738159)  
 GeneralizedExtremeValue{Float64}(μ=9.769433494025204, σ=2.4378484427749645, ξ=0.07852578361203225)  
 GeneralizedExtremeValue{Float64}(μ=23.98917341875042, σ=1.5363662009549794, ξ=0.16874967454442893)  
 GeneralizedExtremeValue{Float64}(μ=9.658300869215411, σ=0.8469496299050212, ξ=0.13829340411875893)

Calculating the discount rate: Here I have chosen to keep 0.04 and 0.02 as they correspond to the mean discount rate and standard deviation discount rate, respectively, as previously seen in Lab04.

function draw\_discount\_rate()  
 return rand(Normal(0.04, 0.02))   
end

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

1. Define an illustrative action, SOW, and model parameters, and run a simulation.

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

ModelParams(House{Int64}(672, 500000, 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  
 0.0  
 15.0  
 18.0  
 20.0  
 23.0  
 25.0  
 27.0  
 28.0  
 30.0  
 ⋮  
 58.0  
 64.0  
 68.0  
 72.0  
 76.0  
 80.0  
 84.0  
 88.0  
 92.0  
 96.0  
 100.0  
 100.0), String7("COM8"), "504", String31("USACE - Galveston"), "Cafeteria Restaurant, 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}(16.86212964, 1.432395797, -0.002082949, 2070.663411, 25.16966141), GeneralizedExtremeValue{Float64}(μ=28.89862382234145, σ=0.5492177973169725, ξ=0.08738699883931017), 0.027995339199131317)

a = Action(3.0u"ft")  
res = run\_sim(a, sow, p)

-1.4332046747786012e7

## 2.2 Large ensemble

1. Sample many SOWs (see below)

sows = [SOW(rand(slr\_scenarios), draw\_surge\_distribution(), draw\_discount\_rate()) for \_ in 1:10] # for 10 SOWs  
actions = [Action(3.0u"ft") for \_ in 1:10] # these are all the same  
results = [run\_sim(a, s, p) for (a, s) in zip(actions, sows)]

10-element Vector{Float64}:  
 -1.761548346466175e6  
 -9.919005367585203e6  
 -4.077562360508768e6  
 -5.19751521182907e6  
 -1.1577242491408994e7  
 -8.509040261396708e6  
 -75609.76800000001  
 -6.322922429031029e6  
 -7.554792153539233e6  
 -75622.21169706853

1. Sample a range of actions. You can do this randomly, or you can look at just a couple of actions (e.g., 0, 3, 6, 9, 12 ft) – explain your choice. I chose to run this loop for the no elevation (0ft), as well as for 5ft, 8ft, and 14ft. While 14ft is not a very realistic actions, including this in the simulation helps contextualize the results for the other values. The 5ft and 8ft made the most sense when looking at the elevation-cost plot above (question 1.c. under “Exploratory Modeling”).
2. Run the simulations for each SOW and action. You can use a for loop for this.

all\_actions = [Action(0.0u"ft"), Action(5.0u"ft"), Action(8.0u"ft"), Action(14.0u"ft")]  
for a in all\_actions  
 print(a)  
 sows = [SOW(rand(slr\_scenarios), draw\_surge\_distribution(), draw\_discount\_rate()) for \_ in 1:10] # for 10 SOWs  
 actions = [a for \_ in 1:10] # these are all the same  
 results = [run\_sim(a, s, p) for (a, s) in zip(actions, sows)]  
 print(mean(results))  
end

Action{Float64}(0.0)-6.026731253894236e6Action{Float64}(5.0)-6.014004446292987e6Action{Float64}(8.0)-6.500073266059774e6Action{Float64}(14.0)-1.5253890396564214e6

1. Create a DataFrame of your key inputs and results (see below)

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],  
)

## 2.3 Analysis

Some questions to consider:

* When do you get the best results? The best results occurred when I set the action to 8ft.
* When do you get the worst results? I had the worst results when I set the action to 0ft.
* What are the most important parameters? The most important parameters are the house elevation (height in feet) and the discount rate because they can be easily measured and therefore easier to change in the model. p
* If you had unlimited computing power, would you run more simulations? How many? Yes, each time I run this simulation the results are different as it is computing the mean of 10 random scenarios. In order to get a better estimate of the results the simulation should be run closer to 10,000 or 100,000 times.
* What are the implications of your results for decision-making? These results can help inform decision-making on how much to elevate the building by and when the optimal time would be to elevate it however, this model should only be used as a resource to help guide the restaurant’s decision. This model is not perfect and is not able to take into account all of the uncertainties that will affect the final decision.