Lab 5: Sea-Level Rise

Jonah Schaechter js336>

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

Choosing Galveston Pier 21, Texas The guage is at 29° 18.6 N, 94° 47.6 W https://maps.app.goo.gl/GyanSMA2fp9rkVrT9

Our building is 302 17th St, Galveston, TX 77550, Home area as estimated by google maps: 30ftx50ft home = 1500ft^2 Home value from zillow: 247,700 (Round up to 250,000)

The home is 4.41 feet or 1.34 meters above sea level in elevation. Looking at it on street view, the house appears to be on concrete blocks about 6 inches tall, giving it an effective height of 4.91 feet. Round this up to 5 so that it works.

Row 98 from the data is two-story, no basement in Galveston, so we’ll be using that for our depth-damage curve. The home is on concrete blocks, so we can be confident that it doesn’t have a basement.

house = let  
 haz\_fl\_dept = CSV.read("data/haz\_fl\_dept.csv", DataFrame) # read in the file  
 desc = "Two-story, no basement in Galveston"  
 row = @rsubset(haz\_fl\_dept, :Column1 == 98)[1, :,] # select the row I want  
 area = 1500u"ft^2"  
 height\_above\_gauge = 5u"ft" #Previously used 4.91, had to round up to 5 to make it work with functions  
 House(  
 row;  
 area=area,  
 height\_above\_gauge=height\_above\_gauge,  
 value\_usd=250\_000,  
 )  
end

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

Check we can get the elevation cost:

elevation\_cost(house, 10u"ft")

161370.0

And then plot elevation cost so we can visually make sure that it makes sense:

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

Time to model sea level rise!

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)]  
end  
println("There are $(length(slr\_scenarios)) parameter sets")

Plot data to visually verify that we read it in correctly:

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

Storm surge distribution function. We’re taking a sample around our flood distribution values that we used in the previous labs.

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)

Draw a distribution of storm surge values:

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

1000-element Vector{GeneralizedExtremeValue{Float64}}:  
 GeneralizedExtremeValue{Float64}(μ=3.9391115782149395, σ=1.0728187449442603, ξ=0.11878238347143774)  
 GeneralizedExtremeValue{Float64}(μ=4.116308530832845, σ=0.7790117429943353, ξ=0.0627618933981908)  
 GeneralizedExtremeValue{Float64}(μ=3.693424906026786, σ=0.9196513850960184, ξ=0.22821963899175457)  
 GeneralizedExtremeValue{Float64}(μ=3.2591653610805715, σ=3.202938863835156, ξ=0.15586048827194446)  
 GeneralizedExtremeValue{Float64}(μ=4.0528883452426365, σ=4.092113404204755, ξ=0.11200097367730905)  
 GeneralizedExtremeValue{Float64}(μ=3.905587673493932, σ=0.8235419151573765, ξ=0.07163028853912835)  
 GeneralizedExtremeValue{Float64}(μ=6.16053708676858, σ=0.12679900886241308, ξ=0.22409371329354574)  
 GeneralizedExtremeValue{Float64}(μ=4.4999490642022755, σ=2.3455653686180122, ξ=0.1310275660429217)  
 GeneralizedExtremeValue{Float64}(μ=6.8219747776567985, σ=0.6622289618240781, ξ=0.04926271772466138)  
 GeneralizedExtremeValue{Float64}(μ=5.77345570766529, σ=0.13768875160144914, ξ=0.1152412059050885)  
 GeneralizedExtremeValue{Float64}(μ=6.184500790556516, σ=0.5508690989634369, ξ=0.09017946827598763)  
 GeneralizedExtremeValue{Float64}(μ=3.8983658688890084, σ=0.7230982829433712, ξ=0.032031546755567436)  
 GeneralizedExtremeValue{Float64}(μ=4.9250654052697165, σ=0.23476759510667633, ξ=0.045231371304879725)  
 ⋮  
 GeneralizedExtremeValue{Float64}(μ=4.499878784290201, σ=2.7760768372058213, ξ=0.13793859071313241)  
 GeneralizedExtremeValue{Float64}(μ=5.249198749770773, σ=0.2923079968263801, ξ=0.07378706300148125)  
 GeneralizedExtremeValue{Float64}(μ=6.34679041878525, σ=2.014951452254529, ξ=0.19654800226446553)  
 GeneralizedExtremeValue{Float64}(μ=5.371654178431248, σ=1.034980674064447, ξ=0.032446529298462715)  
 GeneralizedExtremeValue{Float64}(μ=5.520171461210472, σ=0.14806521322904626, ξ=0.1612429632828695)  
 GeneralizedExtremeValue{Float64}(μ=5.271693223087017, σ=1.5734765989515882, ξ=0.045700706421334364)  
 GeneralizedExtremeValue{Float64}(μ=4.895027582386733, σ=0.7581725699051146, ξ=0.15272922696371288)  
 GeneralizedExtremeValue{Float64}(μ=5.422598043462509, σ=0.7340779302225093, ξ=0.020941700242282174)  
 GeneralizedExtremeValue{Float64}(μ=3.8460199827284276, σ=4.415438657804995, ξ=0.1590192231064239)  
 GeneralizedExtremeValue{Float64}(μ=5.432771591998073, σ=0.12014378310671255, ξ=0.036537910141187874)  
 GeneralizedExtremeValue{Float64}(μ=4.9969003163049726, σ=3.504336132319521, ξ=0.1371189390703317)  
 GeneralizedExtremeValue{Float64}(μ=4.385332378248131, σ=2.2344205341834136, ξ=0.12371636410275652)

Get some discount rate samples. I’m choosing to keep the sampling as-is here.

Justfication: Current USA interest rates are at 5% and I personally think they’ll stay high for a while (I can exlain my reasoning for this but idk if it’s on-topic for this lab), but I’m compensating for the fact that I don’t believe assuming that people will be richer in the near future in Galveston (Speaking as someone who grew up in one, tourism-based economies are just as liable to shrink as they are to grow), so I’m taking it back down from 5% to 4% to compensate for this.

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

Model the house from 2024 to 2083. This is much longer than I did in the past, but I wanted the high-variance of sea level rise in the long term to come in to play.

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

Get SOW from surge distribution, discount rate, and sea level rise scenarios

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

Establish an action:

a = Action(3.0u"ft")

Find Net Present Value:

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

-2.1248770706224986e6

Column\_length = 10  
# I'm not well caught up on predictions or galveston weather, so I feel most comfortable sampling randomly for SLR and storm surge.   
#For discount rate, I'd say I'm confident enough in my knowledge of macroeconomics to choose a distribution, but not enough to bias that distribution in any given direction.  
sows = [SOW(rand(slr\_scenarios), draw\_surge\_distribution(), draw\_discount\_rate()) for \_ in 1:Column\_length]   
  
#For situations like these where sample sizes are less than 10, and randomness or uncertainty aren't part of how high we elevate our house, I believe having a structured set of actions that we can reliable compare to be the best choice  
  
  
actions = [Action(3.0u"ft") for \_ in 1:Column\_length]  
#actions = vcat([Action(0.0u"ft") for \_ in 1:5],[Action(1.0u"ft") for \_ in 1:5])  
results = [run\_sim(a, s, p) for (a, s) in zip(actions, sows)]

10-element Vector{Float64}:  
 -1.4524262257544033e6  
 -152792.77656321676  
 -221799.77472691098  
 -260083.26547275868  
 -723622.1901792922  
 -627548.4676084921  
 -207786.1590938165  
 -294118.790840782  
 -150408.81730853193  
 -143957.4255182403

Make a data frame with our data:

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],  
)

For a constant action of elevating by 3 feet, let’s look at how discount rate impacts our net present value:

let  
 Discount\_Rate = df.discount\_rate  
 NPV = (df.npv)./1\_000\_000  
 Height = Int.(ustrip.(df.Δh\_ft))  
 #have a label for each increment of elevation  
 #labels = ["$i feet of elevation" for i in 0:9]  
 scatter(  
 Discount\_Rate,  
 NPV;  
 color=Height,  
 xlabel="Discount Rate",  
 ylabel="NPV (Millions of USD)",  
 #group=labels,#repeat('0':'9', 1),  
 #label = ["a" "b"],#[i] for i in 0:9,  
 label="NPV given discount rate for 3ft elevation",  
 legend=:bottomright,  
 size=(800, 400),  
 yformatter=:plain, # prevents scientific notation  
 )  
end

There looks like there’s some visible impact. Does this hold for more samples? What happens if we do 100?

function Get\_Samples(Height, Col)  
 sows = [SOW(rand(slr\_scenarios), draw\_surge\_distribution(), draw\_discount\_rate()) for \_ in 1:Col]   
 actions = [Action(Height\*u"ft") for \_ in 1:Col]  
 results = [run\_sim(a, s, p) for (a, s) in zip(actions, sows)]  
  
 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],  
)  
 return df  
end

Get\_Samples (generic function with 1 method)

df = Get\_Samples(3, 100)  
let  
 Discount\_Rate = df.discount\_rate  
 NPV = (df.npv)./1\_000\_000  
 Height = Int.(ustrip.(df.Δh\_ft))  
 scatter(  
 Discount\_Rate,  
 NPV;  
 color=Height,  
 xlabel="Discount Rate",  
 ylabel="NPV (Millions of USD)",  
 label="NPV given discount rate for 3ft elevation",  
 legend=:bottomright,  
 size=(800, 400),  
 yformatter=:plain, # prevents scientific notation  
 )  
end

Things look a little different now. While there is still a correlation between discount rate and npv, it’s more visible in the bottom part of the graph. Many NPVs seem to be higher regardless of the discount rate. This is probably due to some flooding scenarios not being as bad as others. If there is little flooding, NPV will be less dependant on discount rate, because future costs are less of a concern.

We want to compare outcomes for different actions, so let’s do that.

samples = 10  
df = Get\_Samples(0, samples)   
Discount\_Rate = df.discount\_rate  
NPV = (df.npv)./1\_000\_000  
Height = Int.(ustrip.(df.Δh\_ft)[1])  
scatter(  
 Discount\_Rate,  
 NPV;  
 color=Height,  
 xlabel="Discount Rate",  
 ylabel="NPV (Millions of USD)",  
 label="NPV given discount rate for $Height ft elevation",  
)  
  
#c = 1 #color variable  
for i in [1, 4, 8, 12]  
 df = Get\_Samples(i, samples)   
 Discount\_Rate = df.discount\_rate  
 NPV = (df.npv)./1\_000\_000  
 Height = Int.(ustrip.(df.Δh\_ft)[1])  
 println(Height)  
 display(scatter!(  
 Discount\_Rate,  
 NPV;  
 palette = :tab10, #color pallette that's easier to tell apart  
 color=Height,   
 xlabel="Discount Rate",  
 ylabel="NPV (Millions of USD)",  
 label="NPV given discount rate for $Height ft elevation",  
 ))  
 #c += 1  
 #return current()  
end

1  
4  
8  
12

Our results are very interesting! It looks like raising the house higher increases NPV, but it also decreases the dependence of the NPV on the discount rate. This makes sense, because the higher the house is elevated, the less it will flood in the future, and the less impact future losses have on NPV.

Now lets look at how NPV depends on the storm surge mean:

samples = 10  
df = Get\_Samples(0, samples)   
SurgeMean = df.surge\_μ  
NPV = (df.npv)./1\_000\_000  
Height = Int.(ustrip.(df.Δh\_ft)[1])  
scatter(  
 SurgeMean,  
 NPV;  
 color=Height,  
 xlabel="Storm Surge Mean (ft)",  
 ylabel="NPV (Millions of USD)",  
 label="NPV given Storm Surge Mean for $Height ft elevation",  
)  
  
#c = 1 #color variable  
for i in [1, 4, 8, 12]  
 df = Get\_Samples(i, samples)   
 SurgeMean = df.surge\_μ  
 NPV = (df.npv)./1\_000\_000  
 Height = Int.(ustrip.(df.Δh\_ft)[1])  
 println(Height)  
 display(scatter!(  
 SurgeMean,  
 NPV;  
 palette = :tab10, #color pallette that's easier to tell apart  
 color=Height,   
 xlabel="Storm Surge mean (ft)",  
 ylabel="NPV (Millions of USD)",  
 label="NPV given Storm Surge Mean for $Height ft elevation",  
 ))  
 #c += 1  
 #return current()  
end

1  
4  
8  
12

NPV doesn’t seem very dependent on storm surge mean. This is probably becasue we’re looking at this simulation across many years, and floods will eventually happen across these time scales, even in low-surge simulations.

samples = 10  
df = Get\_Samples(0, samples)   
Tstar = df.slr\_tstar  
NPV = (df.npv)./1\_000\_000  
Height = Int.(ustrip.(df.Δh\_ft)[1])  
scatter(  
 Tstar,  
 NPV;  
 color=Height,  
 xlabel="Year that rapid sea level rise commenses",  
 ylabel="NPV (Millions of USD)",  
 label="NPV given T\* for $Height ft elevation",  
)  
  
#c = 1 #color variable  
for i in [1, 4, 8, 12]  
 df = Get\_Samples(i, samples)   
 Tstar = df.slr\_tstar  
 NPV = (df.npv)./1\_000\_000  
 Height = Int.(ustrip.(df.Δh\_ft)[1])  
 println(Height)  
 display(scatter!(  
 Tstar,  
 NPV;  
 palette = :tab10, #color pallette that's easier to tell apart  
 color=Height,   
 xlabel="Year that rapid sea level rise commenses",  
 ylabel="NPV (Millions of USD)",  
 label="NPV given T\* for $Height ft elevation",  
 ))  
 #c += 1  
 #return current()  
end

1  
4  
8  
12

T\* is the year that sea levels begin increasing rapidly in our simulation. There seems to be some dependence of NPV on T\*.

Now for comparison, let’s directly look at the dependence of NPV on how much we elevate the home:

samples = 10  
df = Get\_Samples(0, samples)   
NPV = (df.npv)./1\_000\_000  
Height = Int.(ustrip.(df.Δh\_ft))  
scatter(  
 Height,  
 NPV;  
 color=0,  
 xlabel="Elevation Height",  
 ylabel="NPV (Millions of USD)",  
 #legend=false,  
 label="NPV given 0 ft elevation",  
)  
  
#c = 1 #color variable  
for i in [1, 4, 8, 12]  
 df = Get\_Samples(i, samples)   
 NPV = (df.npv)./1\_000\_000  
 Height = Int.(ustrip.(df.Δh\_ft))  
 #println(Height)  
 display(scatter!(  
 Height,  
 NPV;  
 palette = :tab10, #color pallette that's easier to tell apart  
 color=i,   
 xlabel="Elevation Height",  
 ylabel="NPV (Millions of USD)",  
 #legend=false,  
 label="NPV given $i ft elevation",  
 ))  
 #c += 1  
 #return current()  
end

Of all the variables we have looked at, elevation height has by far the greatest impact on NPV outcomes, and a very clear dependency can be observed.

* When do you get the best results?

For the elevation values tested, we see the best results around 8 feet of elevation. In many scenarios, we start to see some diminishung returns when elevating higher to 12 ft.

* When do you get the worst results?

When not elevating, or when elevating a small amount. This house benefits greatly from being elevated.

* What are the most important parameters?

By far, how high the home is elevated is the most important parameter

* If you had unlimited computing power, would you run more simulations? How many?

Yes. I’d be curious to look at trends in the extreme scenarios, and see what patterns we can find or become clearer with more data points. I’d want at least 100 simulations per home elevation.

* What are the implications of your results for decision-making?

Elevating this home is very important for it’s NPV and how it will fare in the future. Although there is such a thing as elevating so high that it’s not worth it any more, for this home, you have to elevate very high (at least above 8 feet!) to reach that point.