Final Project Report

Grant Parajuli (gap6)

Tue., Apr. 30

Table of contents

[1. Introduction 1](#_Toc165320599)

[1.1 Problem Statement 1](#_Toc165320600)

[1.2 Selected Feature 2](#_Toc165320601)

[2. Literature Review 2](#_Toc165320602)

[3. Methodology 3](#_Toc165320603)

[3.1 Implementation 3](#_Toc165320604)

[3.2 Validation 4](#_Toc165320605)

[4. Results 8](#_Toc165320606)

[5. Conclusions 10](#_Toc165320607)

[5.1 Discussion 10](#_Toc165320608)

[5.2 Conclusions 11](#_Toc165320609)

[6. References 11](#_Toc165320610)

# 1. Introduction

## 1.1 Problem Statement

In our house elevation problem, we have considered approaches with a build policy that does not take into account new information over the course of the simulation. However, we expect future changes in storm surge and sea level rise due to climate change to be important non-stationary variables that may warrant deviation from a static build policy when discovering new information. Our current model only considers a static build policy, which is not representative of how people may decide whether to elevate their house or not given new information. A more realistic approach may consider the “present state” water level over the simulation and use that information to inform the decision of whether or not to elevate a house.

## 1.2 Selected Feature

I plan to implement an approach similar to Garner and Keller (2018) to simulate a build policy using a buffer height and freeboard height that respond to the current state of water level. I use an optimization approach with the buffer height and freeboard height as my decision variables and Net Present Value (NPV) as an objective function. A buffer height defines the acceptable minimum height of the house over the expected water level, while the freeboard height also adds some extra height to avoid the need to elevate constantly. My approach thus uses the current state of the expected water level to inform if an elevation is necessary. It’s possible this might improve over a prescriptive elevation approach, but most importantly, it shows how a different decision-making process affects our outcomes.

# 2. Literature Review

Dynamic approaches to problems have shown they can improve performance when compared to static policies. We studied this in class with the parking garage example, where a solution that built more garage levels based on the projected demand was possibly more profitable than a fixed-build solution. Neufville, Scholtes, and Wang (2006) This same process has extended to environmental applications, such as in water resources planning. Herman et al. (2020) Dynamic approaches allow us to deal with our future uncertainty by using new information to inform our decisions. In Garner and Keller (2018), a dynamic approach was applied to the issue of optimizing the build policy for elevating a dike. I implemented a similar process for our house elevation problem to both use an approach more similar to real-world decision-making and to possibly improve our performance on cost savings.

The approach of Garner and Keller (2018) minimizes 2 objective functions: one for the investment cost of elevating our house and one for the expected flooding damages. Using 2 objective functions gives a sense of the tradeoff between investment costs and damages, but I used only NPV for simplicity of comparison with our previous lab work. Additionally, the paper used several decision variables with a polynomial fit for its buffer and freeboard height policies that may be able to change over each time step. While this is a better approach than mine, I implemented an approach using only the buffer height and freeboard height as decision variables. This still means that the current water level will affect the decision to elevate a house, but the policy itself will not change over the course of the simulation based on the rate of sea level rise. A more complex analysis would consider these features.

# 3. Methodology

## 3.1 Implementation

All of the modifications are in HouseElevation. First, I added a SeqAction struct in core.jl to describe the buffer and freeboard height policies that will be used in a sequential run. Next, I added a run\_sim\_seq function that takes in a SeqAction struct and determines the NPV of a sequential decision policy. This code snippet is below, with an explanation following.

"""  
Run the sequential decision model for a given SeqAction and SOW  
"""  
function run\_sim\_seq(a::SeqAction, sow::SOW, p::ModelParams)  
 # get an expected storm surge value  
 storm\_surges\_ft = range(  
 quantile(sow.surge\_dist, 0.0005); stop=quantile(sow.surge\_dist, 0.9995), length=130  
 )  
 pdf\_values = pdf.(sow.surge\_dist, storm\_surges\_ft) # probability of each  
 exp\_surge = sum(storm\_surges\_ft .\* pdf\_values)/sum(pdf\_values) # weighted average to use for heightening heuristic  
 house\_height = p.house.height\_above\_gauge\_ft  
  
 eads = map(p.years) do year   
 # first, we need to determine the actual elevation height based on our policy  
 slr\_ft = sow.slr(year)  
 exp\_depth = slr\_ft + exp\_surge  
 # only heighten if we expect to be below the buffer height  
 if a.buff < house\_height - exp\_depth  
 Δh = 0  
 else  
 Δh = exp\_depth - (house\_height - a.buff) + a.free  
 # note that we can't elevate more than 14 feet  
 if Δh > 14.0  
 Δh = 14.0  
 end  
 end  
 construction\_cost = elevation\_cost(p.house, Δh)  
 depth\_ft\_gauge = storm\_surges\_ft .+ slr\_ft # flood at gauge including uncertainty  
 # now calculate damages for our heightening  
 depth\_ft\_house = depth\_ft\_gauge .- (house\_height += Δh) # flood @ house, updating height  
 damages\_frac = p.house.ddf.(depth\_ft\_house) ./ 100 # damage   
 weighted\_damages = damages\_frac .\* pdf\_values # weighted damage  
 # Trapezoidal integration of weighted damages  
 ead = trapz(storm\_surges\_ft, weighted\_damages) \* p.house.value\_usd  
 # each element of eads will now be a tuple of the form (damage, construction cost)  
  
 tup = tuple(ead, construction\_cost)  
 end  
 # we need to unzip the array of tuples into an array of EAD and construction costs  
 eads, costs = (first.(eads), last.(eads))  
 years\_idx = p.years .- minimum(p.years)  
 discount\_fracs = (1 - sow.discount\_rate) .^ years\_idx  
 ead\_npv = sum(eads .\* discount\_fracs)  
 costs\_npv = sum(costs .\* discount\_fracs)  
 return -(ead\_npv + costs\_npv)  
end

First, I calculate the expected value for storm surge using the same samples from the storm surge distribution used in the original run\_sim. Over each timestep of the simulation, this value is added to the sea level rise to get the expected water level for that time step. I then compare the difference between the height of the house and the expected water level with the buffer height. If the difference does not exceed the buffer height, then there is no elevation at that time step. If the buffer height is exceeded, the heightening is equal to the freeboard height + the elevation required to reach the buffer height. Afterwards, run\_sim updates the construction cost at each timestep based on the elevation required and then calculates the expected flood damage. After iterating through all these timesteps, the discount rate is applied to all construction costs and expected annual damages and the NPV is returned.

Additionally, in our original implementation, it was not allowed to elevate more than 14 feet. If a buffer/freeboard policy recommended a combined elevation of over 14 feet, I overrode the heightening to be 14 feet.

## 3.2 Validation

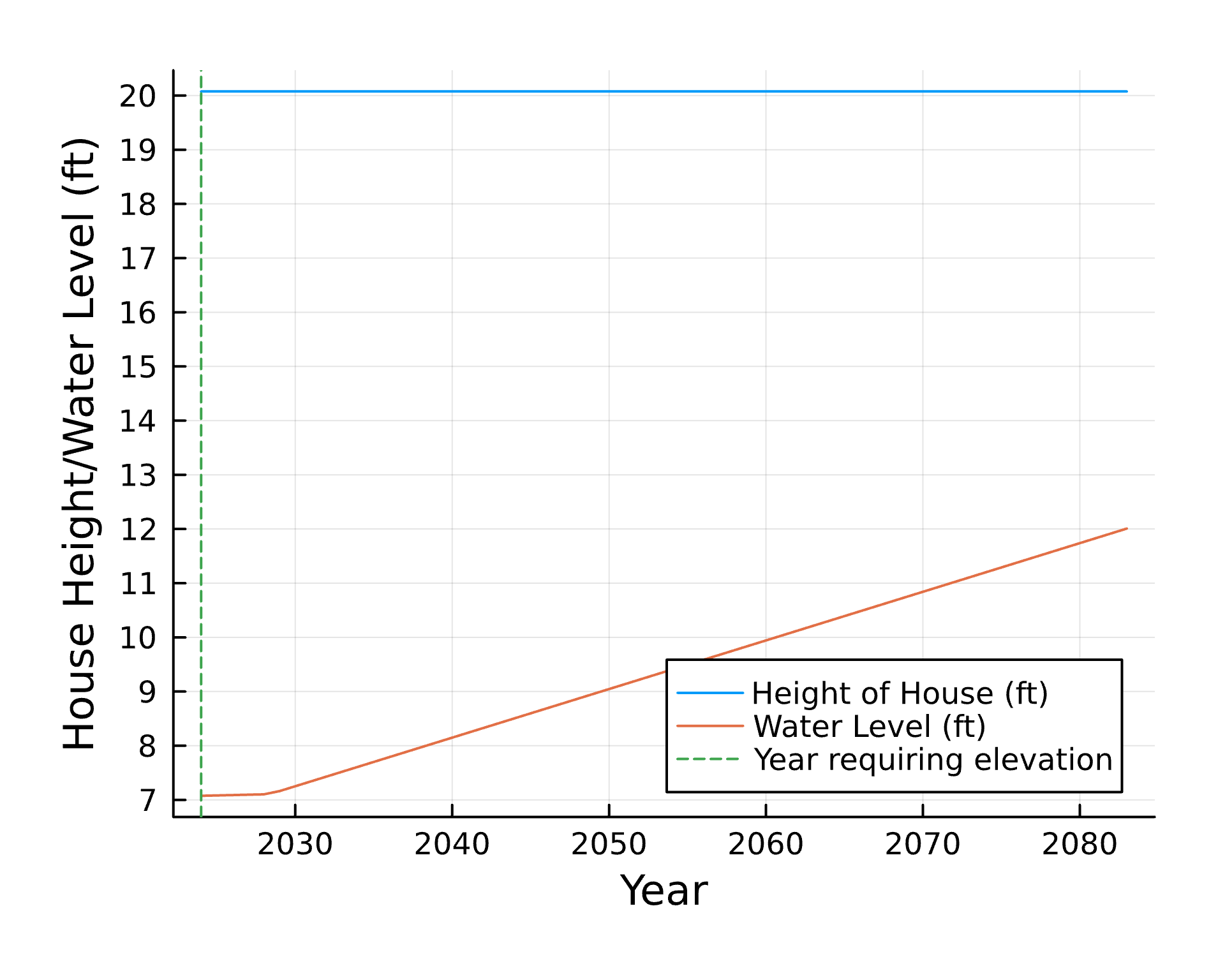
house = let  
 haz\_fl\_dept = CSV.read("data/haz\_fl\_dept.csv", DataFrame) # read in the file  
 desc = "one story, Contents, fresh water, short duration"  
 row = @rsubset(haz\_fl\_dept, :Description == desc)[1, :] # select the row I want  
 area = 500u"ft^2"  
 height\_above\_gauge = 12u"ft"  
 House(row; area=area, height\_above\_gauge=height\_above\_gauge, value\_usd=250\_000)  
end  
  
# define SOWs  
  
p = ModelParams(; house=house, years=2024:2083)  
  
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  
  
function draw\_surge\_distribution()  
 μ = rand(Normal(5, 1))  
 σ = rand(Exponential(1.25))  
 ξ = rand(Normal(0.1, 0.05))  
 return GeneralizedExtremeValue(μ, σ, ξ)  
end  
  
function draw\_discount\_rate()  
 return rand(Normal(0.05, 0.03))  
end  
  
function draw\_sow()  
 slr = rand(slr\_scenarios)  
 surge\_params = draw\_surge\_distribution()  
 discount = draw\_discount\_rate()  
 return SOW(slr, surge\_params, discount)  
end

To validate run\_sim\_seq, I edited it to print out the decision to elevate and the updated elevations of the house for each iteration. I initially found a bug that caused me to reset the elevation on each iteration, and I fixed this. I then plotted the elevations determined by run\_sim\_seq with the expected water level for a random SOW to make sure it made sense with the chosen policy. To do this, I made an altered run\_sim\_seq that returns an array containing tuples with the elevation changes made each year and the water level each year.

function run\_sim\_seq\_elev(a::SeqAction, sow::SOW, p::ModelParams)  
 # get an expected storm surge value  
 storm\_surges\_ft = range(  
 quantile(sow.surge\_dist, 0.0005); stop=quantile(sow.surge\_dist, 0.9995), length=130  
 )  
 pdf\_values = pdf.(sow.surge\_dist, storm\_surges\_ft) # probability of each  
 exp\_surge = sum(storm\_surges\_ft .\* pdf\_values)/sum(pdf\_values) # weighted average to use for heightening heuristic  
 house\_height = p.house.height\_above\_gauge\_ft  
  
 annual\_height\_water = map(p.years) do year   
 # first, we need to determine the actual elevation height based on our policy  
 slr\_ft = sow.slr(year)  
 exp\_depth = slr\_ft + exp\_surge  
 # only heighten if we expect to be below the buffer height  
 if a.buff < house\_height - exp\_depth  
 Δh = 0  
 else  
 Δh = exp\_depth - (house\_height - a.buff) + a.free  
 # note that we can't elevate more than 14 feet  
 if Δh > 14.0  
 Δh = 14.0  
 end  
 end  
 house\_height += Δh  
 tup = tuple(house\_height, exp\_depth)  
 end  
 return annual\_height\_water  
end

I used this function to plot how the house elevation changes with expected water level. The vertical line is placed where the buffer height requirement is violated. As you can see from the plot, this is indeed where the model function predicts that elevation is required. For a 12 foot house with a 5 ft buffer height, a water level of 7 ft or higher requires elevation, which we do here. I tested this for several different random SOWs and sequential actions, and it performed correctly for all of them.

# sample SOWs, both for validation and future optimization  
Random.seed!(987413) # random seed for repeatability  
N\_SOW = 10\_000  
N\_SOW\_opt = 10 # to start  
sows = [draw\_sow() for \_ in 1:N\_SOW]  
sows\_opt = first(sows, N\_SOW\_opt)



Minimal validation was done on the NPV calculation part of run\_sim\_seq, as it used the same approach as in run\_sim. The only minor adjustment was that costs happen on a per year basis depending on whether or not we elevate. I added this functionality and discounted the costs appropriately. Some print statements ensured that costs are accrued only when a house elevation occurs. I also tested different random seeds when running the actual optimization functions, and there was little difference between results.

# 4. Results

I compared the results from running optimization on my method and optimization on the method from lab 6.

# setup optimization  
  
# decision vars  
bounds\_seq = boxconstraints(; lb=[0.0, 0.0], ub=[14.0, 14.0])  
  
# objective function  
function objective\_function\_seq(heights::Vector{Float64})  
 a = SeqAction(heights)  
 npvs = [run\_sim\_seq(a, sow, p) for sow in sows\_opt]  
 return -mean(npvs)  
end  
  
bounds\_og = boxconstraints(; lb=[0.0], ub=[14.0])  
# original objective function for comparison  
function objective\_function\_og(Δh::Vector{Float64})  
 a = Action(Δh[1])  
 npvs = [run\_sim(a, sow, p) for sow in sows\_opt]  
 return -mean(npvs)  
end

objective\_function\_og (generic function with 1 method)

# optimize over many SOW  
N\_SOW\_opt = 100  
sows\_opt = first(sows, N\_SOW\_opt)  
options = Options(; time\_limit=300.0, f\_tol\_rel=10.0)  
algorithm = ECA(; options=options)

Algorithm Parameters  
====================  
 ECA(η\_max=2.0, K=7, N=0, N\_init=0, p\_exploit=0.95, p\_bin=0.02, ε=0.0, adaptive=false, resize\_population=false)  
  
Optimization Result  
===================  
 Empty status.

result\_seq = optimize(objective\_function\_seq, bounds\_seq, algorithm)

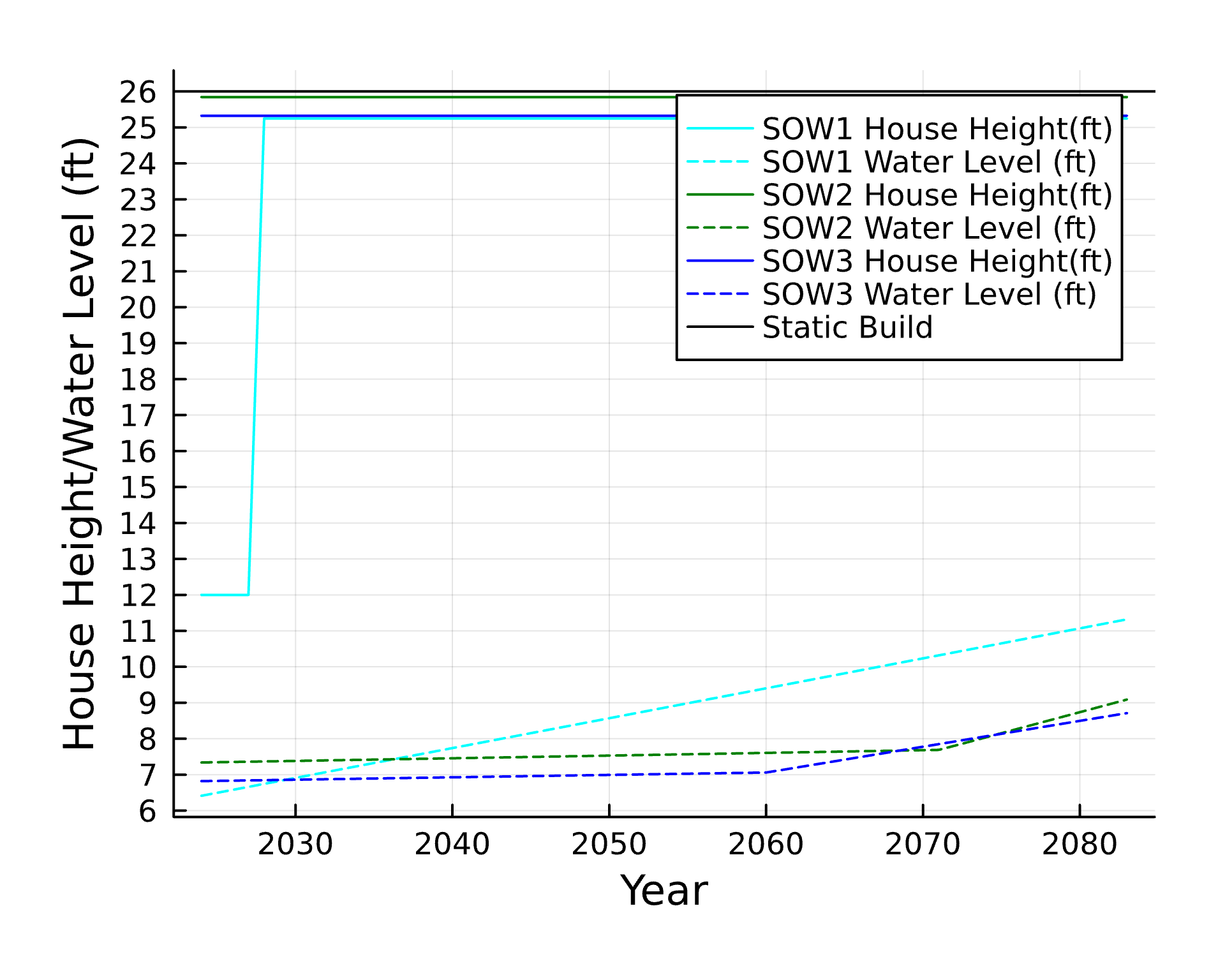
Optimization Result  
===================  
 Iteration: 248  
 Minimum: 75228.3  
 Minimizer: [5.32606, 13.1781]  
 Function calls: 3472  
 Total time: 86.4694 s  
 Stop reason: Due to Convergence Termination criterion.

The optimal buffer height was found to be 5.32604ft, and the optimal freeboard height was found to be 13.1781ft. These resulted in an NPV of -$75,228.3.

# need to repeat these for some reason when optimizing again  
options = Options(; time\_limit=180.0, f\_tol\_rel=10.0)  
algorithm = ECA(; options=options)  
result\_og = optimize(objective\_function\_og, bounds\_og, algorithm)  
# 80637.5

Optimization Result  
===================  
 Iteration: 67  
 Minimum: 96203  
 Minimizer: [14.0]  
 Function calls: 469  
 Total time: 32.5990 s  
 Stop reason: Due to Convergence Termination criterion.

The optimal fixed build policy was to elevate by 14 feet, with an NPV of -$96,203. The plot below compares this fixed policy with the sequential decision policy over 3 SOWs.



Water levels are dashed, while house heights are solid. The solid black line at the top represents the static build policy.

# 5. Conclusions

## 5.1 Discussion

There is a small improvement in using the sequential decision policy over the static policy over the SOWs I studied. While my policy often continues to recommend high elevations, it also does this by responding to the current water level, allowing the potential for a delayed or lower heightening if water levels do not rise too high. By delaying heightening, future costs become discounted, so this is one possible savings source. Due to the high freeboard height, only one elevation is also generally required, which is why these policies end up being somewhat comparable.

One major caveat to the sequential decision method implemented here is that it does not have any reliability criteria. The method implemented in Garner and Keller (2018) was more sophisticated and required reliability in 80% of SOWs. This is an important constraint as delaying a heightening may be a bad decision if it affects reliability.

As always, I would benefit from optimizing over more SOWs. Additionally, I assume that the weighted average storm surge is a good approximation for the storm surge in each year to use in the heightening heuristic. This implicitly assumes stationarity in storm surge, limiting some of the ability of this approach to adapt to flexibility. Since we’ve seen in many systems that reliance on historical data may result in policies that quickly become out-dated, an improvement on this work would be to include the non-stationarity of storm surge over the course of the simulation as this would affect our expectations of water levels.

My approach also only used NPV as an objective function and only optimized the static policy for a buffer height and freeboard height. A more sophisticated approach would include the ability to change these policies at each time step, weighting sea level rise as in Garner and Keller (2018). Additionally, splitting NPV into construction costs and flood damages would result in an exploration of trade-offs between the two. In this project, I only used NPV as an objective function to more easily compare with our previous optimization of NPV.

## 5.2 Conclusions

My project found that implementing an approach that decides whether to elevate a house based on the current water level can result in more cost savings compared to a static build policy when comparing optimal solutions. This is also a more realistic decision policy as home-owners may base their choice to elevate their house on the current state of water levels affected by sea-level rise. This approach can save costs by delaying elevation expenses or elevating less than the static build policy.

Some important caveats to this work are the assumption that expected storm surge is stationary. I also do not include a reliability constraint, which would improve my approach in different SOWs. Additionally, more complex models can include the trade-offs between investment costs and damages and also allow the buffer and freeboard height policies to vary with the expected changes in sea level rise. Overall, this work suggests that a sequential decision process may be better than static build policies and that further improvements on risk assessment should consider such strategies.

# 6. References

Garner, Gregory G., and Klaus Keller. 2018. “Using Direct Policy Search to Identify Robust Strategies in Adapting to Uncertain Sea-Level Rise and Storm Surge.” *Environmental Modelling & Software* 107 (September): 96–104. <https://doi.org/10.1016/j.envsoft.2018.05.006>.

Herman, Jonathan D., Julianne D. Quinn, Scott Steinschneider, Matteo Giuliani, and Sarah Fletcher. 2020. “Climate Adaptation as a Control Problem: Review and Perspectives on Dynamic Water Resources Planning Under Uncertainty.” *Water Resources Research*, January, e24389. <https://doi.org/10.1029/2019wr025502>.

Neufville, Richard de, Stefan Scholtes, and Tao Wang. 2006. “Real Options by Spreadsheet: Parking Garage Case Example.” *Journal of Infrastructure Systems* 12 (2): 107–11. <https://doi.org/10.1061/(asce)1076-0342(2006)12:2(107)>.