Final Project Report

Jonah Schaechter (JS336)

Tue., Apr. 30

1 Introduction

1.1 Problem Statement

So far, our simulations have considered the Net Present Value of home elevation under the assumption that homeowners will pay for their home elevation out of pocket. In reality, people have to chose between paying out of pocket (if this is even something they can afford), taking out a loan, or saving up.

Each of these options has tradeoffs in total cost, yearly cost, the cost of damages,and in NPV. In this paper, we will analize these tradeoffs.

1.2 Selected Feature

Describe the feature you have selected to add to the existing decision-support tool. Discuss how this feature relates to the problem statement and its potential to improve climate risk assessment.

2 Literature Review

Provide a brief overview of the theoretical background related to your chosen feature. Cite at least two relevant journal articles to support your approach (see Quarto docs for help with citations). Explain how these articles contribute to the justification of your selected feature.

3 Methodology

3.1 Implementation

```
import Pkg
Pkg.status()  # Note the paths for your local packages
Pkg.gc()  # Garbage collect and delete cached package files
```

```
Status `C:\Users\Jonah\OneDrive\Documents\Rice PhD\Climate Damage Mitigation\final-project-js33
  [336ed68f] CSV v0.10.13
  [a93c6f00] DataFrames v1.6.1
  [1313f7d8] DataFramesMeta v0.15.2
  [31c24e10] Distributions v0.25.107
  [49e40289] HouseElevation v0.1.0 `HouseElevation`
```

```
  [7073ff75] IJulia v1.24.2
  [b964fa9f] LaTeXStrings v1.3.1
  [bcdb8e00] Metaheuristics v3.3.5
  [b6ca2b7d] ParkingGarage v0.1.0 `ParkingGarage`
  [91a5bcdd] Plots v1.40.2
  [295af30f] Revise v3.5.14
  [2913bbd2] StatsBase v0.34.3
  [1986cc42] Unitful v1.19.0
  [9a3f8284] Random
Info Packages marked with  have new versions available and may be upgradable.
```

```
      Active manifest files: 10 found
      Active artifact files: 121 found
      Active scratchspaces: 2 found
     Deleted no artifacts, repos, packages or scratchspaces
```

```
using CSV
using DataFrames
using DataFramesMeta
using Distributions
using LaTeXStrings
using Metaheuristics
using Plots
using Random
using Unitful

Plots.default(; margin=5Plots.mm)
```

```
using Revise
using HouseElevation
#include("Finance.jl")
#using house
```

We put in our house in galveston

Choosing Galveston Pier 21, Texas The guage is at 29° 18.6 N, 94° 47.6 W https://maps.app.goo.gl/GyanSMA2fp9r

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 = height_above_gauge = 5u"ft"
    House(row; area=area, height_above_gauge=height_above_gauge, value_usd=250_000)
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

function draw_surge_distribution()
      = rand(Normal(5, 1))
      = rand(Exponential(1.25))
      = rand(Normal(0.1, 0.05))
    return GeneralizedExtremeValue( ,  ,  )
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
```

Generate our possible states of the world

```
Random.seed!(421521)
N_SOW = 10
N_SOW_opt = 50 # to start
sows = [draw_sow() for _ in 1:N_SOW]
sows_opt = first(sows, N_SOW_opt)
```

```
10-element Vector{SOW{Float64}}:
 SOW{Float64}(Oddo17SLR{Float64}(58.66628415, 2.709557857, 0.003600726, 2061.196955, 24.7855783
 SOW{Float64}(Oddo17SLR{Float64}(18.98238754, 2.076689405, 0.002459112, 2017.216294, 15.5840157
 SOW{Float64}(Oddo17SLR{Float64}(13.54334493, 1.576769557, -0.001892038, 2019.630551, 13.563066
 SOW{Float64}(Oddo17SLR{Float64}(38.14589083, 2.623948719, 0.005085519, 2016.508542, 8.1346290
 SOW{Float64}(Oddo17SLR{Float64}(40.9256403, 2.858600773, 0.008671168, 2024.544892, 18.7522310
 SOW{Float64}(Oddo17SLR{Float64}(36.12774556, 2.258399655, 0.002399946, 2066.78925, 19.7923147
 SOW{Float64}(Oddo17SLR{Float64}(54.44686357, 2.440676913, 0.003209864, 2039.323894, 22.8970514
 SOW{Float64}(Oddo17SLR{Float64}(42.54886307, 2.374354031, 0.003786949, 2018.717192, 9.5870908
 SOW{Float64}(Oddo17SLR{Float64}(5.951503134, 1.501882544, -0.002460307, 2022.058777, 10.78953
 SOW{Float64}(Oddo17SLR{Float64}(17.66182918, 1.54415968, -0.002055399, 2058.500619, 31.6412588
```

Write our function we'll use to evaluate financial plans

#“‘{julia}

p = ModelParams(; house=house, years=2024:2083, finance=finance_basic) #make our model

elevations_try = 0:0.5:14 #establish actions to try actions_try = Action.(elevations_try)

#Get all the contruction costs we can afford construction_costs = [elevation_cost(house, elevation) for elevation in elevations_try if elevation_cost(house, elevation) <= 170_000]

actions_try = actions_try[1:length(construction_costs)] #get the list of actions we can afford

#“‘

3.2 Financial_NPVS function

```julia
#assume physical house and SLR scenarios stay constant, only financial conditions change
function financial_npvs(finance)
    p = ModelParams(; house=house, years=2024:2083, finance=finance)  #make our model

    elevations_try = 0:0.5:14  #establish actions to try
    actions_try = Action.(elevations_try)

    #Get all the contruction costs we can afford
    construction_costs = [elevation_cost(house, elevation) for elevation in elevations_try if e

    actions_try = actions_try[1:length(construction_costs)]  #get the list of actions we can a
    elevations_try = elevations_try[1:length(construction_costs)]  #limit elevations as well s

    #Run simulations
    npvs_opt = [mean([run_sim(a, sow, p) for sow in sows_opt]) for a in actions_try]

    #get height that minimizes NPV
    min_npv, min_idx = findmax(npvs_opt)
    minimizer = elevations_try[min_idx]

    #return elevations we tried (x), npvs(y) and minimizing elevation
    return elevations_try, npvs_opt, minimizer

end
```

```
financial_npvs (generic function with 1 method)
```

Basic financing, just paying up fronts

println(methods(Finance)) println(fieldtypes(Finance)) ## Finance_Basic

```julia
finance_basic = let
    loan = 0
    loan_years = 0
    loan_rate = 0.0
    paid_off_percent = 1.0
    amnt_paid_off = paid_off_percent * house.value_usd  # Calculate amnt_paid_off
```

```
    # Create Finance object using keyword arguments
    Finance(;
        loan = loan,
        loan_years = loan_years,
        loan_rate = loan_rate,
        paid_off_percent = paid_off_percent,
        amnt_paid_off = amnt_paid_off
    )
end
```

```
Finance(0, 0, 0.0, 1.0, 250000.0)
```

Now lets calculate our basic finance options

```
elevations_tried, npvs, min = financial_npvs(finance_basic)
```

```
(0.0:0.5:14.0, [-1.0914604085744503e6, -1.0905425222436262e6, -953694.5075908549, -824837.0348
```

3.3 plot_many

Define our function to plot many financial scenarios

```
function plot_many(finance_list, initial_plot)


    for finance in finance_list

        #println(finance)
        elevations_tried, npvs, min = financial_npvs(finance)
        if finance.loan == 1  #if taking out a loan
            label = "$(round(finance.loan_rate*100))% loan over $(finance.loan_years) years"
        elseif finance.loan > 1
            label = "Saving over $(finance.loan_years) years"
        else
            label = "Paying out of pocket"
        end
        plot!(
        initial_plot,
        elevations_tried,
        npvs ./ 1000;
        xlabel="Elevation [ft]",
        ylabel="NPV [1000 USD]",
        label=label,
        marker=:circle,
        #color=line_color
        )

         #make the vertical line color the same as the horizontal line
        line_color = initial_plot.series_list[end][:linecolor]
```

```
        #line_color = plot_object.series_list[end].plotseries[:linecolor]
        vline!([min]; label="$(min)ft elevation", linestyle=:dash, color=line_color)
    end
end
```

```
plot_many (generic function with 1 method)
```

3.4 reasonable saving time

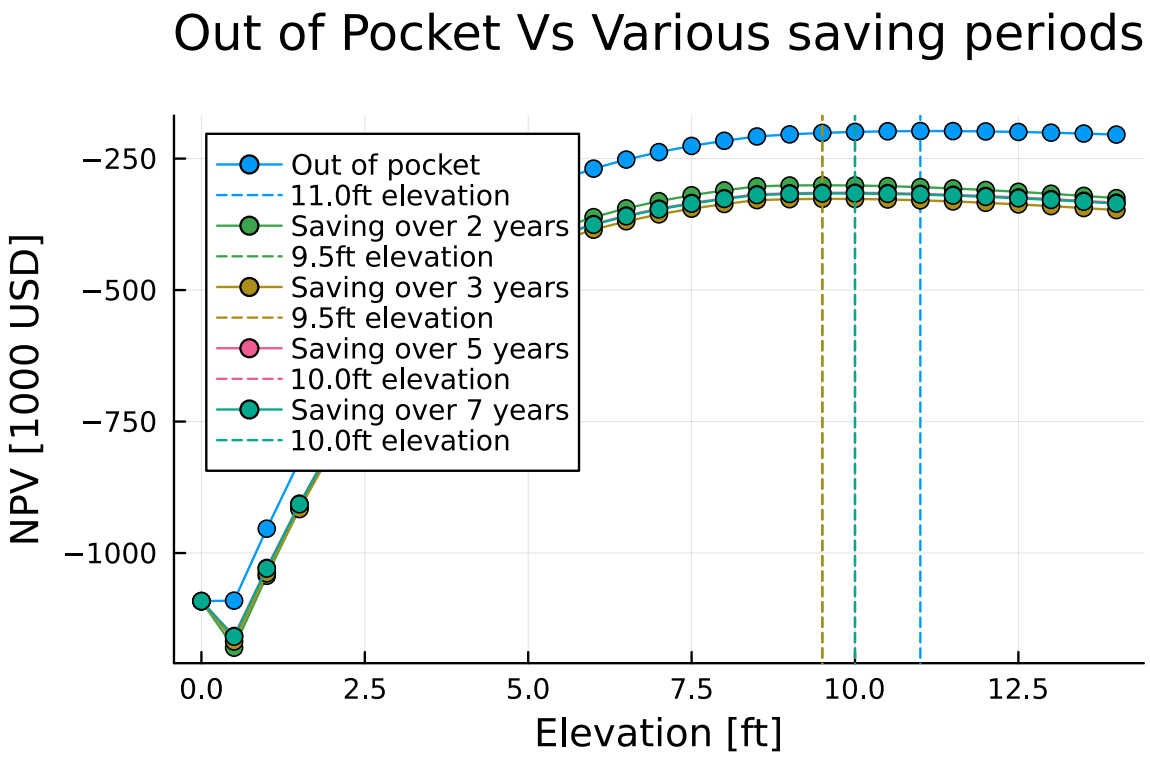
Generate many financial scenarios where we save up for different periods of time

```
finance_saving = []

for i in [2, 3, 5, 7]
    #print(i)
    paid_off_percent = 1.0
    fin = Finance(; loan=2, loan_years=i, loan_rate=0.0, paid_off_percent=paid_off_percent,
        amnt_paid_off=(house.value_usd*paid_off_percent)
        )
    push!(finance_saving, fin)
end

println(finance_saving)
```

```
Any[Finance(2, 2, 0.0, 1.0, 250000.0), Finance(2, 3, 0.0, 1.0, 250000.0), Finance(2, 5, 0.0, 1
```

Plot our saving options

```
p = plot(
    elevations_tried,
    npvs ./ 1000;
    xlabel="Elevation [ft]",
    ylabel="NPV [1000 USD]",
    title="Out of Pocket Vs Various saving periods",
    label="Out of pocket",
    marker=:circle#, color=line_color
)


line_color = p.series_list[end][:linecolor]  #make the vertical line color the same as the hor
vline!([min]; label="$(min)ft elevation", linestyle=:dash, color=line_color)

plot_many(finance_saving, p)
display(p)
```



Results of this graph show two interesting things:

1. NPV of saving is always a lot lower than paying up front. This makes sense, since we're increasing construction cost with inflation, so the only change to the NPV will be more damages while we wait to elevate the house.
2. NPV results recommend lower elevations if we chose to wait. This seems strange at first, since most people might save money so they can elevate higher, and you'd think that if we do wait to elevate, it'd make more sense to do a higher elevation to compensate for the damages we experienced in the first few years.

But since we are doing this simulation over a fixed period of time, the longer we wait to elevate, the less value is gained by elevating, since we'll have spent less of our time avoiding damages. So assuming a non-infinite time duration for this experiment, the longer we wait to elevate, the lower the recommended elevation height gets.

We can demonstrate this more clearly by displaying some extremely long saving periods.

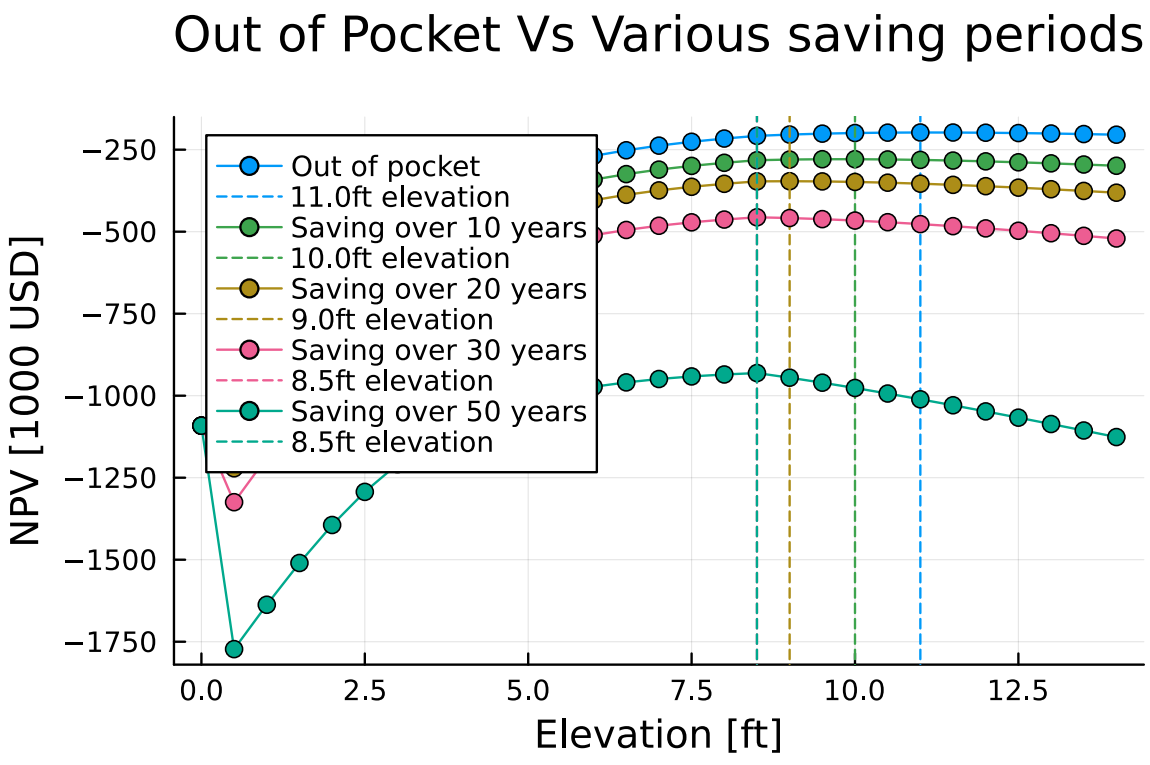
3.5 Long saving time

```
Long_Savings = []

for i in [10, 20, 30, 50]
    #print(i)
    paid_off_percent = 1.0
    fin = Finance(; loan=2, loan_years=i, loan_rate=0.0, paid_off_percent=paid_off_percent,
        amnt_paid_off=(house.value_usd*paid_off_percent)
```

```
        )
    push!(Long_Savings, fin)
end

println(Long_Savings)
```

```
Any[Finance(2, 10, 0.0, 1.0, 250000.0), Finance(2, 20, 0.0, 1.0, 250000.0), Finance(2, 30, 0.0
```

```
p = plot(
    elevations_tried,
    npvs ./ 1000;
    xlabel="Elevation [ft]",
    ylabel="NPV [1000 USD]",
    title="Out of Pocket Vs Various saving periods",
    label="Out of pocket",
    marker=:circle#, color=line_color
)


line_color = p.series_list[end][:linecolor]  #make the vertical line color the same as the hori
vline!([min]; label="$(min)ft elevation", linestyle=:dash, color=line_color)

plot_many(Long_Savings, p)
display(p)
```



3.6 Different loan durations

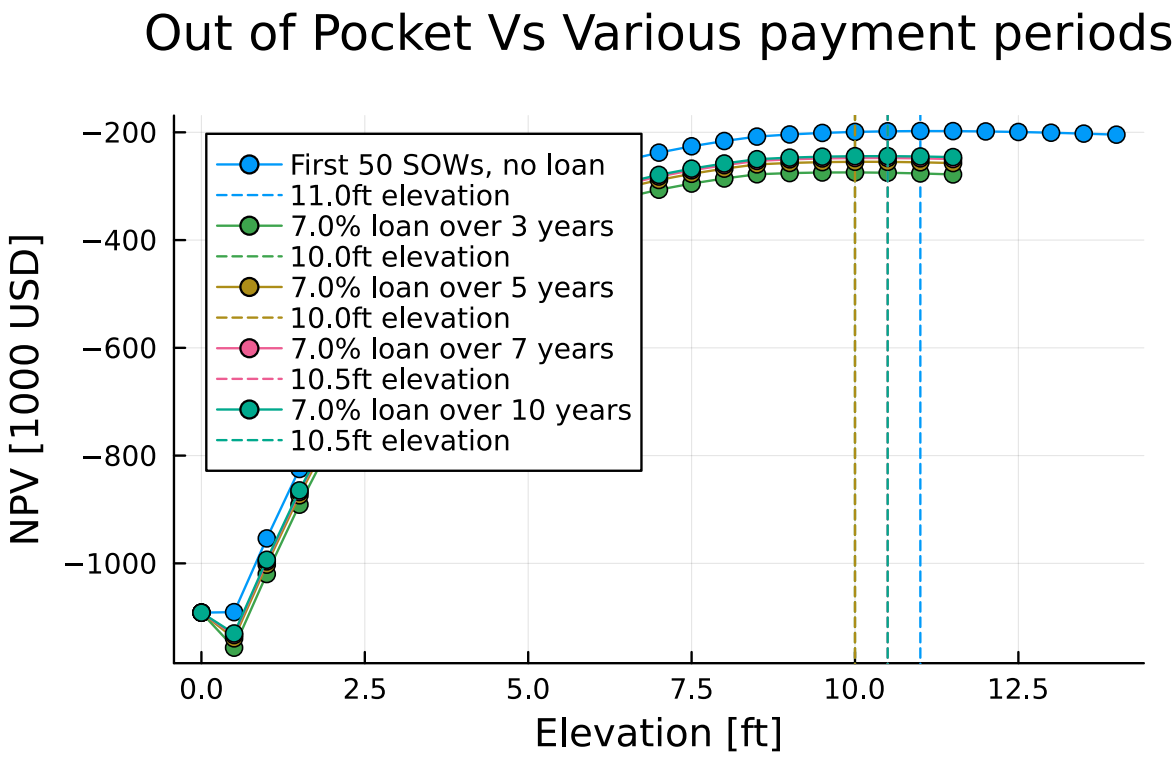
Generate many financial scenarios with different years to pay off the loan at a 7% interest rate

```
finance_list_years = []

for i in [3, 5, 7, 10]
    #print(i)
    paid_off_percent = 0.7
    fin = Finance(; loan=1, loan_years=i, loan_rate=0.07, paid_off_percent=paid_off_percent,
        amnt_paid_off=(house.value_usd*paid_off_percent)
        )
    push!(finance_list_years, fin)
end

println(finance_list_years)
```

```
Any[Finance(1, 3, 0.07, 0.7, 175000.0), Finance(1, 5, 0.07, 0.7, 175000.0), Finance(1, 7, 0.07
```

Plot our scenarios of different years to pay off the loan

```
p = plot(
    elevations_tried,
    npvs ./ 1000;
    xlabel="Elevation [ft]",
    ylabel="NPV [1000 USD]",
    title="Out of Pocket Vs Various payment periods",
    label="First $(N_SOW_opt) SOWs, no loan",
    marker=:circle#, color=line_color
)


line_color = p.series_list[end][:linecolor]  #make the vertical line color the same as the hori
vline!([min]; label="$(min)ft elevation", linestyle=:dash, color=line_color)

plot_many(finance_list_years, p)
display(p)
```



3.7 Different interest rates

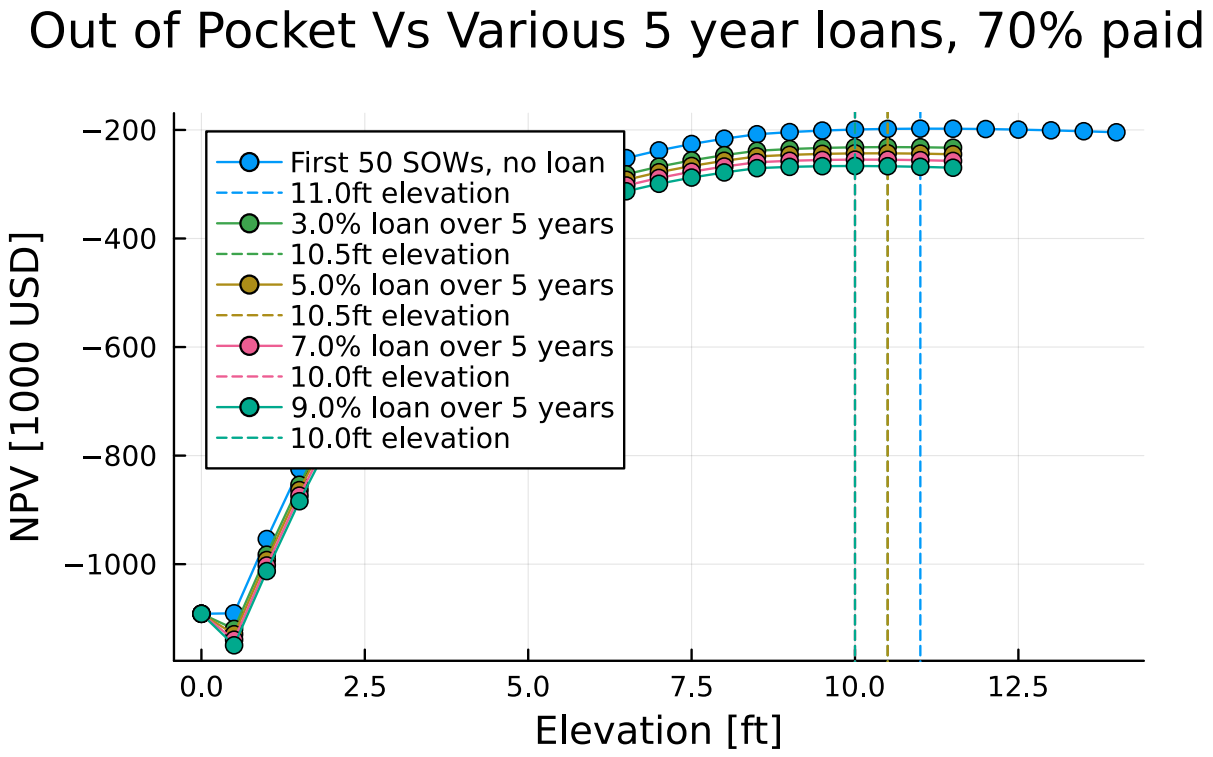
Generate many financial scenarios with different interest rates

```
finance_list_rates = []

for i in range(start=3, stop=9, length=4)
    #print(i)
    paid_off_percent = 0.7
    fin = Finance(; loan=1, loan_years=5, loan_rate=i/100, paid_off_percent=paid_off_percent,
        amnt_paid_off=(house.value_usd*paid_off_percent)
        )
    push!(finance_list_rates, fin)
end

println(finance_list_rates)
```

```
Any[Finance(1, 5, 0.03, 0.7, 175000.0), Finance(1, 5, 0.05, 0.7, 175000.0), Finance(1, 5, 0.07
```

```
#colors = [colorant"red", colorant"green", colorant"blue", colorant"orange", colorant"purple"]

#line_color = colors[1]

p = plot(
    elevations_tried,
    npvs ./ 1000;
```

```
    xlabel="Elevation [ft]",
    ylabel="NPV [1000 USD]",
    title="Out of Pocket Vs Various 5 year loans, 70% paid off",
    label="First $(N_SOW_opt) SOWs, no loan",
    marker=:circle#, color=line_color
)
#println(p.series_list[end][:linecolor] )

#colors = Plots.palette()

line_color = p.series_list[end][:linecolor]  #make the vertical line color the same as the hori
vline!([min]; label="$(min)ft elevation", linestyle=:dash, color=line_color)

plot_many(finance_list_rates, p)
display(p)
```



3.8 Compare NPVS and annual costs

Now we want to compare NPVS and annual costs for all of these scenarios

Make a function to get NPV, annual cost, and optimal elevatin

```
function annual_loan_cost(p, r, n)
    #P is principle amount, r is rate, n is number of years
    a = p * ( r * ((1+r)^n)  / ( (1+r)^n - 1 ) )
    return a #a is annual payments
```

```
end


function annual_v_npv(finance)

    elevations_tried, npvs, best_elevation = financial_npvs(finance)
    #An engineer is not someone who makes things maximally efficient, but someone who makes thi
    best_npv, best_idx = findmax(npvs)

    construction_cost = elevation_cost(house, best_elevation)
    #println(elevation_cost(house, best_elevation))

    if finance.loan == 0 #out of pocket
        annual_cost = construction_cost
    elseif finance.loan == 1  #taking out a loan
        annual_cost = annual_loan_cost(construction_cost, finance.loan_rate, finance.loan_years
    else #saving up
        annual_cost = construction_cost/finance.loan_years
    end

    return best_npv, annual_cost, best_elevation

end
```

```
annual_v_npv (generic function with 1 method)
```

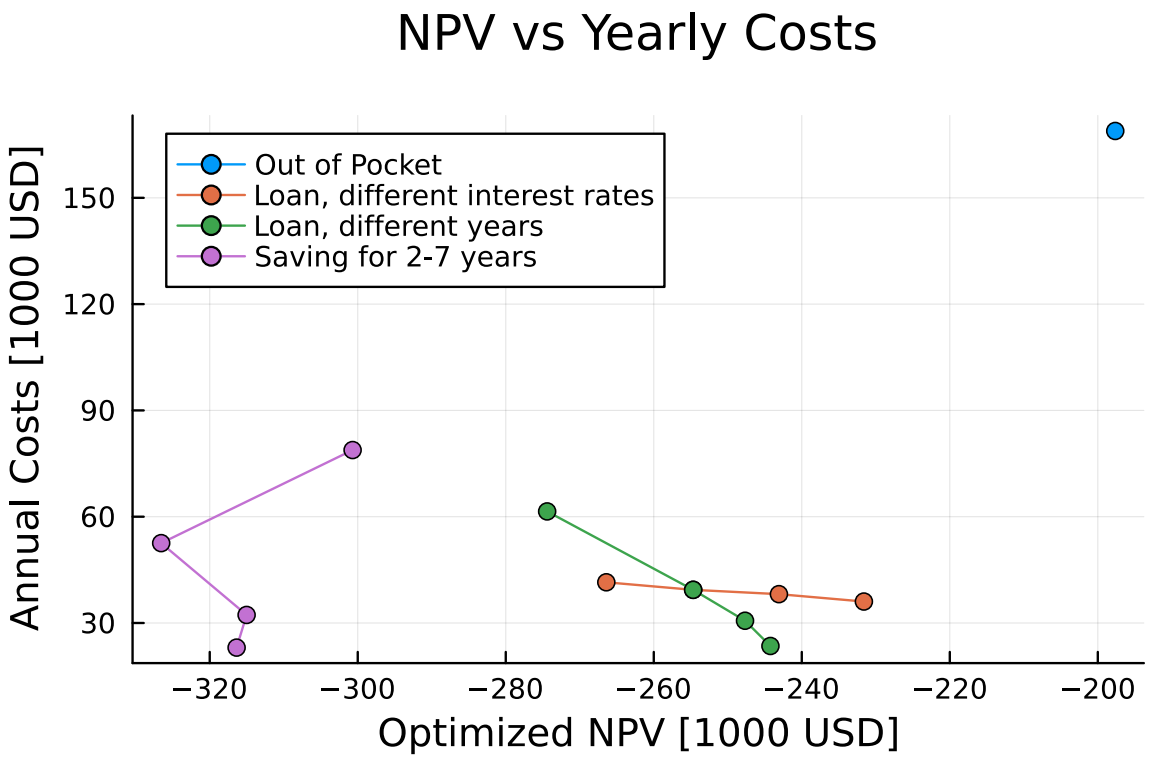
```
function get_costs(finances)
    figures = annual_v_npv.(finances)
    best_npvs = [fig[1] for fig in figures]
    annual_costs = [fig[2] for fig in figures]
    best_elevations = [fig[3] for fig in figures]

    return best_npvs, annual_costs, best_elevations
end


function plot_money_values(finance, initial_plot, label)

    best_npvs, annual_costs, best_elevations = get_costs(finance)
    plot!(
        initial_plot,
        best_npvs ./ 1000,
        annual_costs ./ 1000;
        xlabel="Optimized NPV [1000 USD]",
        ylabel="Annual Costs [1000 USD]",
        label=label,
        marker=:circle,
        #color=line_color
```

```
        )

end
```

```
plot_money_values (generic function with 1 method)
```

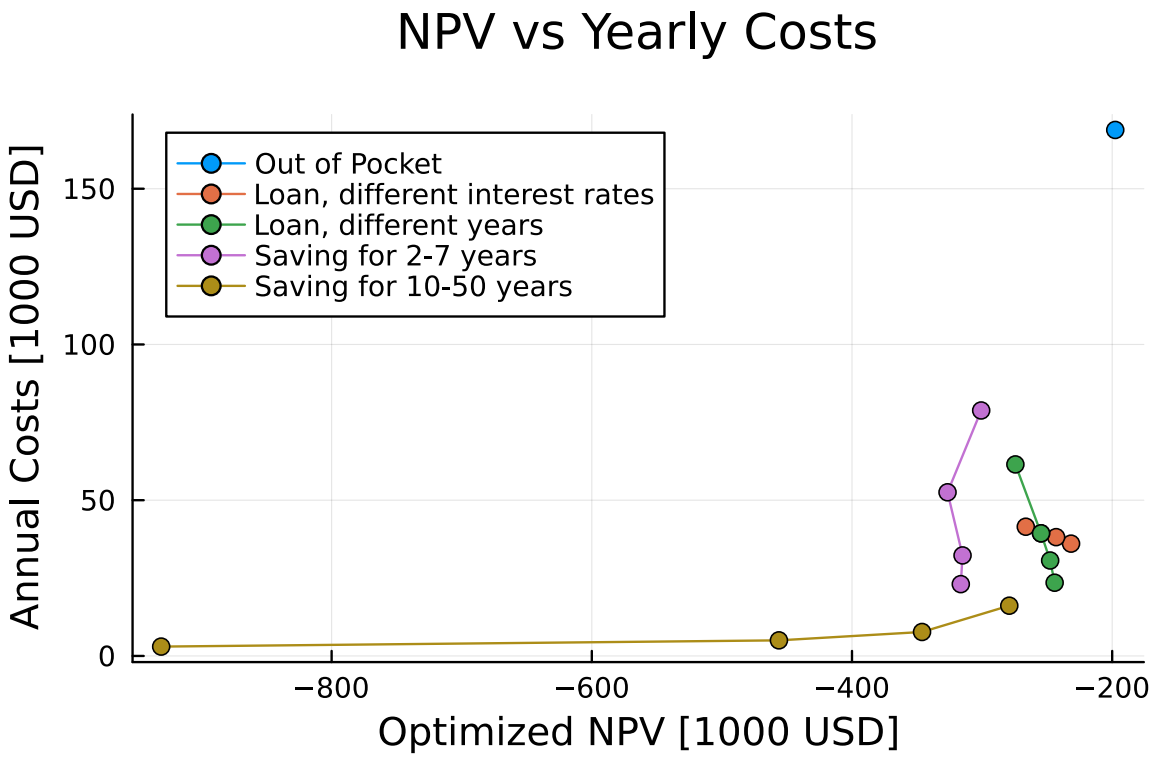
get_costs(finance_list_rates)

```
optimal_npv, annual_cost, best_elevation = annual_v_npv(finance_basic)

Annual_v_npv_plot = plot(
    [optimal_npv] ./ 1000,
    [annual_cost] ./ 1000 ;
    xlabel="Optimized NPV [1000 USD]",
    ylabel="Annual Costs [1000 USD]",
    title="NPV vs Yearly Costs",
    label="Out of Pocket",
    marker=:circle#, color=line_color
)
plot_money_values(finance_list_rates, Annual_v_npv_plot, "Loan, different interest rates")
plot_money_values(finance_list_years, Annual_v_npv_plot, "Loan, different years")
plot_money_values(finance_saving, Annual_v_npv_plot, "Saving for 2-7 years")
display(Annual_v_npv_plot)
```



The chart changes radically when we display long savings times

```
plot_money_values(Long_Savings, Annual_v_npv_plot, "Saving for 10-50 years")
display(Annual_v_npv_plot)
```



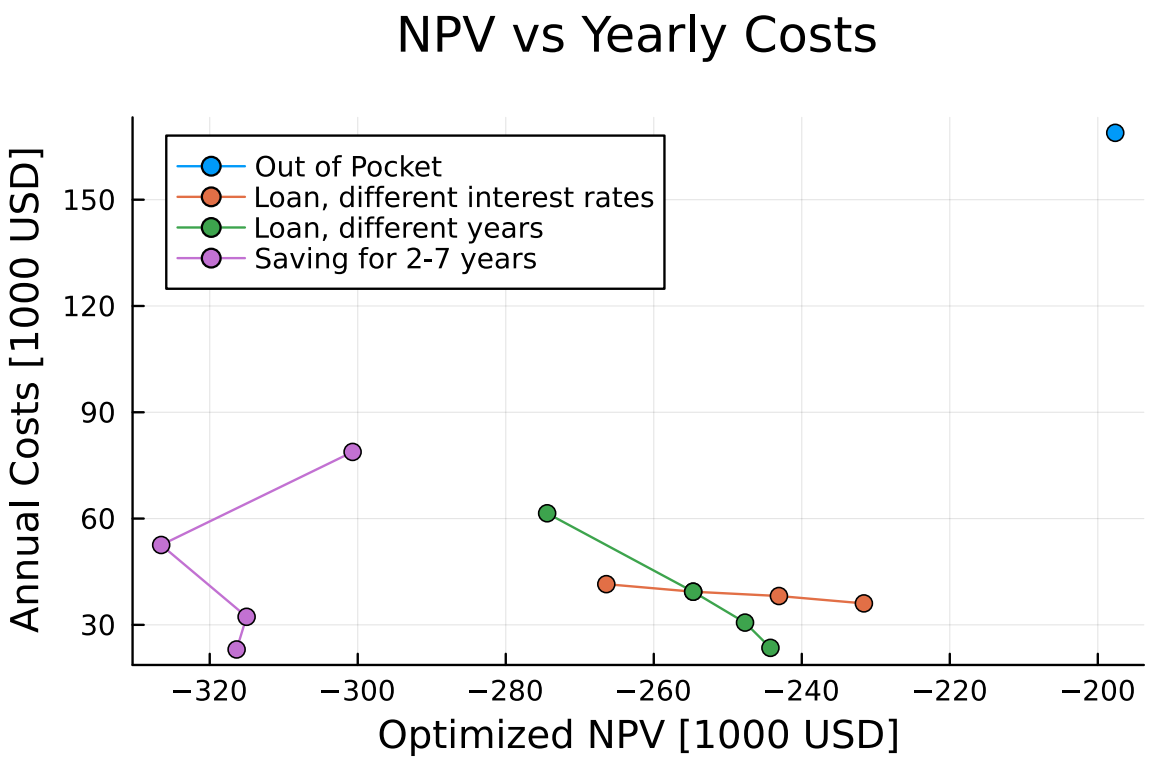
3.9 compare NPVS and construction heights for all these different scenarios

```
function general_plot(x, y, finance, initial_plot, label)

    # 1 = best_npvs, 2 = annual_costs, 3 = best_elevation
    values = collect(get_costs(finance)) #get our values to plot.
    #Using collect makes sure these values are in an array so we can use ./1000 later
    #println(values)
    #println(values[2])
    values[1] = values[1] ./ 1000  #adjust npv and annual costs
    values[2] = values[2] ./1000


    plot!(
        initial_plot,
        values[x],  #decide what to plot
        values[y];
        label=label,
        marker=:circle,
        #color=line_color
        )
```

```
end
```

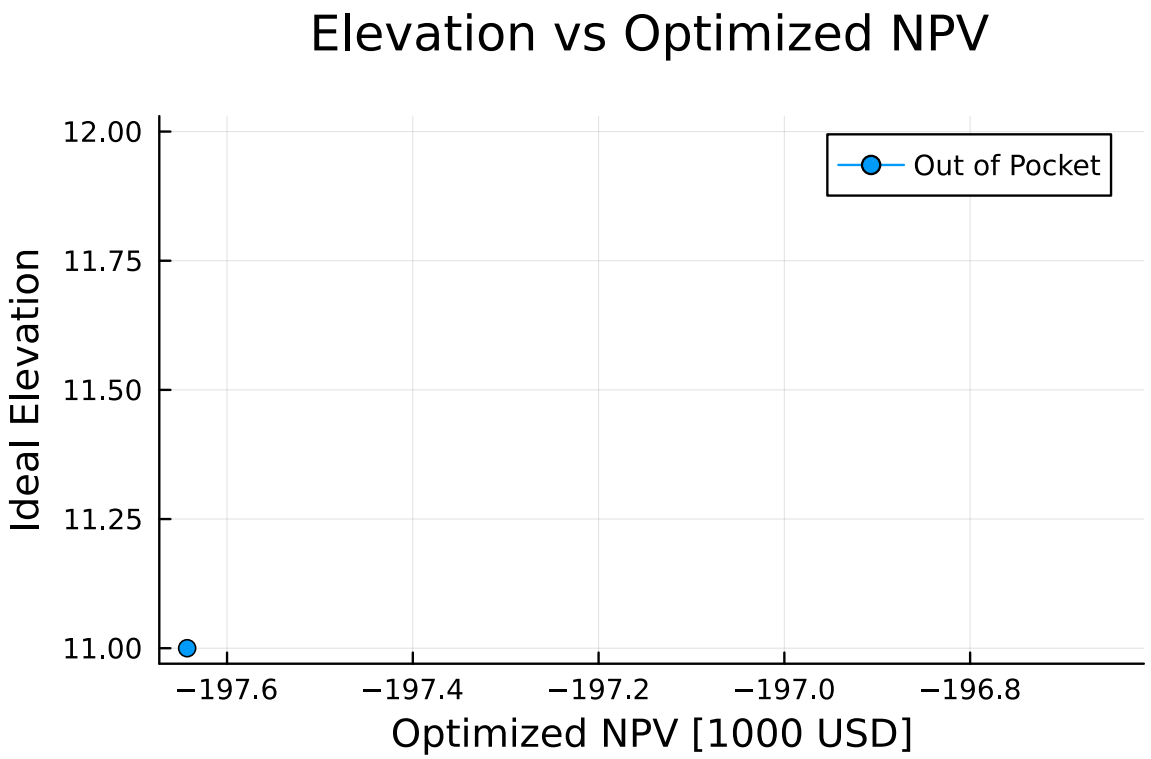
```
general_plot (generic function with 1 method)
```

```
test_plot = plot(
    [optimal_npv] ./ 1000,
    [annual_cost] ./ 1000 ;
    xlabel="Optimized NPV [1000 USD]",
    ylabel="Annual Costs [1000 USD]",
    title="NPV vs Yearly Costs",
    label="Out of Pocket",
    marker=:circle#, color=line_color
)
general_plot(1, 2, finance_list_rates, test_plot, "Loan, different interest rates")
general_plot(1, 2, finance_list_years, test_plot, "Loan, different years")
general_plot(1, 2, finance_saving, test_plot, "Saving for 2-7 years")
#display(Annual_v_npv_plot)
```




```
Annual_v_npv_plot = plot(
    [optimal_npv] ./ 1000,
    [best_elevation] ;
    xlabel="Optimized NPV [1000 USD]",
    ylabel="Ideal Elevation",
    title="Elevation vs Optimized NPV",
    label="Out of Pocket",
    marker=:circle#, color=line_color
```

```
)
```



Now we want to compare NPVS and construction heights for all these different scenarios

3.10 Next: add indicators to annual_v_npv plot that shows what thing is what

You should make your modifications in either the `HouseElevation` or `ParkingGarage` module. Detail the steps taken to implement the selected feature and integrate it into the decision-support tool. Include code snippets and explanations where necessary to clarify the implementation process.

3.11 Validation

In this case, we don't really *need* optimization – we can use brute force. We can compare by plotting the objective function for a range of elevations (from 0 to 14 ft) using all SOWs.

As we have seen in labs, mistakes are inevitable and can lead to misleading results. To minimize the risk of errors making their way into final results, it is essential to validate the implemented feature. Describe the validation techniques used to ensure the accuracy and reliability of your implemented feature. Discuss any challenges faced during the validation process and how they were addressed.

4 Results

Present the results obtained from the enhanced decision-support tool. Use tables, figures, and visualizations to clearly communicate the outcomes. Provide sufficient detail to demonstrate how the implemented feature addresses the problem statement. Use the `#| output: false` and/or `#| echo: false` tags to hide code output and code cells in the final report except where showing the

output (e.g.g, a plot) or the code (e.g., how you are sampling SOWs) adds value to the discussion. You may have multiple subsections of results, which you can create using `##`.

5 Conclusions

5.1 Discussion

Analyze the implications of your results for climate risk management. Consider the context of the class themes and discuss how your findings contribute to the understanding of climate risk assessment. Identify any limitations of your approach and suggest potential improvements for future work.

5.2 Conclusions

Summarize the key findings of your project and reiterate the significance of your implemented feature in addressing the problem statement. Discuss the broader implications of your work for climate risk management and the potential for further research in this area.

6 References