Lab 6: Policy Search

Leanh Nguyen (Ln14)

Fri., Mar. 1

Table of contents

using Revise  
using HouseElevation  
  
using CSV  
using DataFrames  
using DataFramesMeta  
using Distributions  
using LaTeXStrings  
using Metaheuristics  
using Plots  
using Random  
using Unitful  
  
Plots.default(; margin=5Plots.mm)

# 1. Site Information

## 1.1 Pick a site

Galveston Pier 21, TX

## 1.2 Pick a building

Building: Ocean Star Offshore Drilling Rig and Museum Gift Store

Location: 29.310430907645863, -94.79173653184932

Address: 2002 Wharf Rd, Galveston, TX 77550

Reason: This building is very close to the gauge as the gauge is located right outside the building. The gauge is located right between the coast and the building.

## 1.3 Find the building’s elevation

Elevation: roof = 5 meters above sea level, base = 2 meters above sea level

# 2. Problem description

In this lab, we will implement single-objective policy search for our house elevation problem. These methods can also be used for multi-objective policy search, but this does increase computational complexity.

## 2.1 Decision variables

We’re going to focus on a single decision variable: how high to elevate a house. Of course, running a full optimization here is probably overkill, as Zarekarizi, Srikrishnan, and Keller (2020) showed that a brute force search over all possible elevations is sufficient to find a good solution. However, we want to build up some optimization expertise to help us with more complex problems in the future.

We will use a continuous decision variable, the height of the house above the ground. We limit it between 0 and 14 feet.

## 2.2 Objective function

### 2.2.1 Setting Initial Parameters

For now, we’ll keep the same objective function that we’ve been using: net present value, considering both the cost of heightening the house and the discounted expected costs of future flood damage.

As you know, it’s not enough to state the objective function, however. We also need to consider the state(s) of the world over which we will optimize.

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  
  
house = let  
 haz\_fl\_dept = CSV.read("data/haz\_fl\_dept.csv", DataFrame) # read in the file  
 desc = "Gift Shop, structure"  
 row = @rsubset(haz\_fl\_dept, :Description == desc)[1, :] # select the row I want  
 area = 795u"ft^2"  
 height\_above\_gauge = 6u"ft"  
 House(row; area=area, height\_above\_gauge=height\_above\_gauge, value\_usd=250\_000)  
end  
  
p = ModelParams(; house=house, years=2024:2083)  
  
function draw\_surge\_distribution()  
 μ = rand(Normal(5, 1))  
 σ = rand(Exponential(1.5))  
 ξ = rand(Normal(0.1, 0.05))  
 return GeneralizedExtremeValue(μ, σ, ξ)  
end  
function draw\_discount\_rate()  
 return 0.0  
end  
  
N\_SOW = 100  
sows = [  
 SOW(rand(slr\_scenarios), draw\_surge\_distribution(), draw\_discount\_rate()) for  
 \_ in 1:N\_SOW  
] # for 10 SOWs

Line 10

Area was obtained from Google Earth’s measurement tools

Line 11

From Google Earth, I calculated/estimated my target structure’s area (795 sq ft). Then, I found a structure near my target structure on zillow that had a similar area (750 sq ft). Thus, I decided to use this as the house structure value. https://www.zillow.com/homedetails/101-21st-St-STE-214-Galveston-TX-77550/2132158257\_zpid/

### 2.2.2 More-efficient storm surge sampling

### 2.2.3 Validation

We can make sure that we get the same thing by comparing our result to a simple Monte Carlo estimate using 25,000 samples. We can also time how long they take using the @time macro (which is actually not a great way to evaluate code – a better approach uses the [BenchmarkTools](https://juliaci.github.io/BenchmarkTools.jl/stable/) package but we won’t go into this here – instead we run use @time the *second* time we call each function, which gives a better estimate than timing it the first time).

### 2.2.4 Metaheuristics.jl

We are attempting to solve a single-objective optimization problem that is likely nonlinear and nonconvex. We will use the [Metaheuristics.jl](https://jmejia8.github.io/Metaheuristics.jl/stable/) package to do this. This package implements a number of optimization algorithms, including genetic algorithms, that are well-suited to this type of problem. Let’s follow a a quick overview from [the docs](https://jmejia8.github.io/Metaheuristics.jl/stable/).

### 2.2.5 Defining and Testing the Objective Function

1. Set your random seed to 2024 so that you always get the same answers when you re-run your code.
2. Generate N\_SOW = 100\_000 sows at random as in the previous lab and/or as in the template code provided above.
3. Pick the first N\_SOW\_opt = 10 of these sows to use for optimization. You can (and should!!) increase this number once you have a working solution, but we’ll use just a few to make sure everything is working.
4. Define an objective function that takes in a single number as an input (the elevation of the house in feet) and returns one objective function (the net present value of the house for that elevation).
   1. Convert the input scalar to an Action
   2. Call run\_sim on each of the N\_SOW\_opt sows and the elevation to get the expected value of the objective function.
   3. Return the negative of the sum of these expected values (since we are minimizing).
5. Test your objective function with a few different elevations to make sure it’s working.
6. Run the optimization with the objective function and see what elevation it recommends.
7. Validate the result by plotting the objective function for a range of elevations (from 0 to 14 ft) using all your SOWs. Is the recommended elevation the minimum? (We’re lucky that in this problem we can compare our optimization solution to a brute-force approach!) If it doesn’t seem to be the minimum:
   1. try increasing N\_SOW\_opt and see if the result changes.
   2. check whether the optimization algorithm is converging

function objective\_function(elevation)  
 # Set number of SOWS and SOWS for optimization  
 N\_SOW = 100\_000  
 N\_SOW\_opt = 10  
  
 # Generate N\_SOW sows at random  
 sows = [SOW(rand(slr\_scenarios), draw\_surge\_distribution(), draw\_discount\_rate()) for \_ in 1:N\_SOW]  
  
 # Pick the first N\_SOW\_opt of these sows to use for optimization  
 sows\_opt = sows[1:N\_SOW\_opt]  
  
 # Define an objective function that takes in a single number (elevation) and returns the net present value  
 objective\_value = 0.0  
 for sow in sows\_opt  
 # Convert elevation to Action with units  
 action = Action(elevation[1]\*1u"ft")  
 # Run simulation to get the expected value of the objective function  
 objective\_value += -run\_sim(action, sow, p)  
 end  
   
 return objective\_value  
end  
# Set random seed  
Random.seed!(2024)  
  
# Call objective\_function with a specific elevation value  
elevation\_value = 8.0 # Example elevation value, you can use any value within the defined range  
result = objective\_function(elevation\_value)  
  
# Display the result  
println("Objective value for elevation $elevation\_value: $result")  
  
# Test the objective function with different elevation values  
elevations\_to\_test = [0.0, 5.0, 10.0, 14.0] # Example elevation values to test  
for elevation\_value in elevations\_to\_test  
 result = objective\_function(elevation\_value)  
 println("Objective value with 100000 SOWS and 10 SOWS to optimize for elevation $elevation\_value: $result")  
end

Objective value for elevation 8.0: 1.1748458394805156e6  
Objective value with 100000 SOWS and 10 SOWS to optimize for elevation 0.0: 8.056997106710862e6  
Objective value with 100000 SOWS and 10 SOWS to optimize for elevation 5.0: 2.7549764557677163e6  
Objective value with 100000 SOWS and 10 SOWS to optimize for elevation 10.0: 1.9421899929547412e6  
Objective value with 100000 SOWS and 10 SOWS to optimize for elevation 14.0: 1.168803276315263e6

# Set random seed  
Random.seed!(2024)  
algorithm = ECA()  
# Define the search space bounds  
bounds = boxconstraints(lb=[0], ub=[14])  
# Run optimization  
result = optimize(objective\_function, bounds, algorithm)  
# Record time to optimize  
@time optimize(objective\_function, bounds, algorithm)  
# View the minimum of the objective function   
minimum(result)  
# Value of the decision variable that achieves that minimum  
minimizer(result)  
recommended\_elevation = minimizer(result)  
println("Recommended elevation: $recommended\_elevation")

Line 3

The ECA algorithm is suggested as a default, so we’ll use that.

379.338238 seconds (2.14 G allocations: 151.693 GiB, 7.10% gc time, 0.05% compilation time)  
Recommended elevation: [7]

100000 SOWS with 10 SOWS for optimization

* 147.257043 seconds (156.11 M allocations: 55.838 GiB, 1.71% gc time)
* Total Time: 05:58.50
* Recommended elevation: [7] ft

#### 2.2.5.1 Graph 1

# Define elevations range  
elevations\_range = range(0.0, stop=14.0, length=100)  
  
# Calculate objective values for each elevation in the range  
objective\_values = [objective\_function(elevation) for elevation in elevations\_range]  
  
# Plot objective function  
plot(elevations\_range, objective\_values, xlabel="Elevation (ft)", ylabel="Objective Value", label="Objective Function", title="Objective Function v. Recommended Elevation 1")  
scatter!([recommended\_elevation], [objective\_function(recommended\_elevation)], label="Recommended Elevation 1", color="red", markersize=5)

function objective\_function(elevation)  
 # Set number of SOWS and SOWS for optimization  
 N\_SOW = 100\_000  
 N\_SOW\_opt = 15  
  
 # Generate N\_SOW sows at random  
 sows = [SOW(rand(slr\_scenarios), draw\_surge\_distribution(), draw\_discount\_rate()) for \_ in 1:N\_SOW]  
  
 # Pick the first N\_SOW\_opt of these sows to use for optimization  
 sows\_opt = sows[1:N\_SOW\_opt]  
  
 # Define an objective function that takes in a single number (elevation) and returns the net present value  
 objective\_value = 0.0  
 for sow in sows\_opt  
 # Convert elevation to Action with units  
 action = Action(elevation[1]\*1u"ft")  
 # Run simulation to get the expected value of the objective function  
 objective\_value += -run\_sim(action, sow, p)  
 end  
   
 return objective\_value  
end  
# Set random seed  
Random.seed!(2024)  
  
# Call objective\_function with a specific elevation value  
elevation\_value = 8.0 # Example elevation value, you can use any value within the defined range  
result = objective\_function(elevation\_value)  
  
# Display the result  
println("Objective value for elevation $elevation\_value: $result")  
  
# Test the objective function with different elevation values  
elevations\_to\_test = [0.0, 5.0, 10.0, 14.0] # Example elevation values to test  
for elevation\_value in elevations\_to\_test  
 result = objective\_function(elevation\_value)  
 println("Objective value with 100000 SOWS and 10 SOWS to optimize for elevation $elevation\_value: $result")  
end

Objective value for elevation 8.0: 2.886597796670527e6  
Objective value with 100000 SOWS and 10 SOWS to optimize for elevation 0.0: 1.1653343541119061e7  
Objective value with 100000 SOWS and 10 SOWS to optimize for elevation 5.0: 5.681954444106545e6  
Objective value with 100000 SOWS and 10 SOWS to optimize for elevation 10.0: 2.467543879293366e6  
Objective value with 100000 SOWS and 10 SOWS to optimize for elevation 14.0: 2.0598063964112701e6

# Set random seed  
Random.seed!(2024)  
algorithm = ECA()  
# Define the search space bounds  
bounds = boxconstraints(lb=[0], ub=[14])  
# Run optimization  
result = optimize(objective\_function, bounds, algorithm)  
# Record time to optimize  
@time optimize(objective\_function, bounds, algorithm)  
# View the minimum of the objective function   
minimum(result)  
# Value of the decision variable that achieves that minimum  
minimizer(result)  
recommended\_elevation\_2 = minimizer(result)  
println("Recommended elevation: $recommended\_elevation\_2")

436.912381 seconds (2.20 G allocations: 179.113 GiB, 4.17% gc time)  
Recommended elevation: [6]

100000 SOWS with 20 SOWS for optimization

* 217.122129 seconds (2.20 G allocations: 179.113 GiB, 6.67% gc time)
* Total Time: 07:27.92
* Recommended elevation: [6] ft

#### 2.2.5.2 Graph 2

# Define elevations range  
elevations\_range = range(0.0, stop=14.0, length=100)  
  
# Calculate objective values for each elevation in the range  
objective\_values = [objective\_function(elevation) for elevation in elevations\_range]  
  
# Plot objective function  
plot(elevations\_range, objective\_values, xlabel="Elevation (ft)", ylabel="Objective Value", label="Objective Function", title="Objective Function v. Recommended Elevation 2")  
scatter!([recommended\_elevation\_2], [objective\_function(recommended\_elevation\_2)], label="Recommended Elevation 2", color="red", markersize=5)

Try increasing N\_SOW\_opt and see if the result changes: YES, N\_SOW\_opt=10 recommended 7 ft while N\_SOW\_opt=15 recommended 6 ft, a small difference that can play a large role in policy making and economic decisions for the home owner.

Check whether the optimization algorithm is converging: YES, in both graphs, there is large variability in the beginning but the high and low ranges begin to converge as elevation increases past the recommended elevation.

Is the recommended elevation the minimum? - For N\_SOW\_opt=10, it is close to the minimum on the graph. Thus, I increased the N\_SOW\_opt to 15. Now, the recommended elevation is the minimum after some reruns with the graph code. It is important to note that I had to run the code for the graph multiple times to get this result, implying uncertainties in the data.

## 2.3 Reflection

Conclude your analysis by reflecting on the following questions

1. How are we framing this problem? What are the decision variables, the objective function, and the states of the world over which we optimize?

* Optimization
  + In this problem, we are optimizing the elevation of a house to minimize the net present value of damages caused by sea-level rise (SLR) and storm surges. In other words, the optimization problem aims to find the optimal elevation of the house that minimizes the expected damages over a range of possible sea-level rise and storm surge scenarios.
* Decision variables
  + The decision variable is the elevation of the house, which determines its height above the ground level.
* Objective function
  + The objective function is defined as the negative of the net present value of damages incurred over a set of scenarios of sea-level rise and storm surges. The objective of this objective function is to minimize this value, with lower damages being preferable.
* States of the World
  + The states of the world over which we optimize include various scenarios of sea-level rise and storm surges. These scenarios are represented by the sows variable, which contains a collection of different combinations of sea-level rise, surge distributions, and/or discount rates.

1. Diggning deeper, we are averaging the objective function computed over a finite number of states of the world. This assumes that they are all drawn from a distribution representing the “true” distribution of states of the world. Is this a good assumption?

* With the assumption that the states of the world (SOWs) are drawn from a distribution representing the “true” distribution of states of the world, one must consider several pros and cons of this strategy/assumption.
  + Advantages:
    - Computational Efficiency: As simulating all possible states of the world would be impractical or computational impossible, averaging over a finite number of samples allows for a practical and efficient approximation.
  + Disadvantages:
    - Bias and Variance: The sample might not accurately represent the “true” distribution of states of the world, leading to bias in the estimated average objective function.
    - Modeling Errors: The accuracy of the model depends on the variance of the objective function across different states. A high variance requires a larger sample size to achieve a reliable average. Thus, the accuracy of the model used to generate SOWs can impact the validity of the assumption. If the model fails to capture significant factors or introduces biases, the observed SOWs may not accurately represent the true distribution.
    - Stationarity: The assumption assumes that the underlying distribution of SOWs remains stationary over time. However, in reality, environmental factors, economic factors, and other factors may create shifts or changes in the distribution over time.
    - Rare Events: Rare/extreme events that have not been observed in the data may occur in the future. These events can have significant impacts but may not be adequately represented in the observed SOWs under this assumption.
  + On the whole, given these considerations, it’s essential to acknowledge the limitations of assuming that observed SOWs are drawn from the true distribution. Sensitivity analysis, scenario planning, and robust optimization techniques can help mitigate the impact of uncertainties and deviations from this assumption. Additionally, updating models with new data can improve the accuracy of predictions and decision-making processes.

1. What’s not being considered in this analysis that might be important?

* There are several important factors that are not considered in the current analysis which may impact the results:
  + Feedback Loops: Feedback loops within the system, where the output of one process affects the input of another, can have significant impacts on outcomes. In other words, in some cases, the analysis assumes that the simulated states are independent and identitically distributed. Thus, ignoring these feedback loops may lead to oversimplified models and suboptimal results.
  + Long-term Trends: The analysis does not explicitly consider the possibility of the system being time-dependent. Instead, the analysis focuses on short-term variations without considering long-term trends. If the system’s behavior changes over time, simulating states from a single time point might not be representative of the long-term performance. Long-term changes in environmental factors, economic factors, and societal behaviors can have significant impacts on the system’s behavior over time. Similarly, the analysis assumes that the underlying distribution of states is stationary. However, in some cases, the distribution can change from external factors, rending the estimated objective function outdated.
  + Policy and Governance: This analysis does not consider the influence of policy decisions and institutional capacities on adaptation and mitigation efforts. Effective policies and governance mechanisms can be crucial for managing risks and enhancing resilience
  + Social Dynamics: This analysis does not consider socioeconomic factors, which can significantly impact vulnerability and exposure. Although this analysis can give an optimal value for elevation, the result may not be applicable to everyone with varying social and economic circumstances, especially low-income communities.
* However, addressing these factors would require a more comprehensive and integrated modeling approach and access to more datasets. In the face of uncertainty and complexity, incorporating the aforementioned factors can improve the robustness and effectiveness of decision-making processes.

Zarekarizi, Mahkameh, Vivek Srikrishnan, and Klaus Keller. 2020. “Neglecting Uncertainties Biases House-Elevation Decisions to Manage Riverine Flood Risks.” *Nature Communications* 11 (1, 1): 5361. <https://doi.org/10.1038/s41467-020-19188-9>.