# Lab 4: House Elevation NPV Analysis

CEVE 421/521

Thu., Feb. 8

Today we're going to explore net present value (NPV) analysis in the context of a semi-realistic case study of house elevation.

In the previous lab, you developed a depth-damage relationship for a coastal structure and assessed how you might adjust the probability distribution of flooding at a nearby gauge to account for the structure's height relative to the gauge. Today, you'll use the same structure to compare the costs and benefits of elevating the structure to reduce the risk of flooding.

## 0.1 Setup

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.

We begin by loading our packages

```
using CSV
using DataFrames
using DataFramesMeta
using Distributions
using Interpolations
using Plots
using StatsPlots
using Unitful

Plots.default(; margin=6Plots.mm)
```

We also leverage functions defined in another file, as before

```
include("depthdamage.jl")
```

# 1 Building the case study

We are developing a decision-support tool to help assess whether to elevate a house in a flood-prone area. We will use net present value (NPV) analysis to compare the costs and benefits of elevating the house. Specifically, we will consider two kinds of costs:

1. The cost of elevating the house, which we incur only in the first year
2. The annual expected costs of flooding, which we will assume are reduced by elevating the house. We can consider this as an annual insurance premium.

An advantage of this framing is that we don't have to model whether a flood occurs in each given year – we only worry about the probability distribution of flooding. A disadvatage of this framing is that we make some unrealistic assumptions, like that if there is a flood it will automatically be restored to the same condition as before. However, the main advantage is that it simplifies our computation a lot and allows us to focus on the NPV analysis itself.

## 1.1 Adding some math in

We have been using the following notation. In each time step, we calculate $u_t(a, \mathbf{s})$, where $a$ is the action (in this case, how high we choose to elevate), and $\mathbf{s}$ is the "state of the world." As noted above, $u_t(a, \mathbf{s}) = -c_{\text{constr}}(a) - \mathbb{E}[c_{\text{damage}}(a, \mathbf{s})]$, where $c_{\text{constr}}(a)$ is the cost of elevating the house and $c_{\text{damage}}(a, \mathbf{s})$ is the expected cost of flooding in state $\mathbf{s}$ after taking action $a$.

For now, let's define our "state of the world" to have two pieces of information: the probability distribution of flooding, expressed as a Generalized Extreme Value distribution, and the discount rate. For now, we'll treat the depth-damage function and the cost of elevating the house as fixed, although we could consider uncertainty.

Now let's build some of these steps out with some code.

## 1.2 Depth-damage function

In the previous lab, we used existing data to build a depth-data function. Remember that the depth here is relative to the house, not to the gauge.

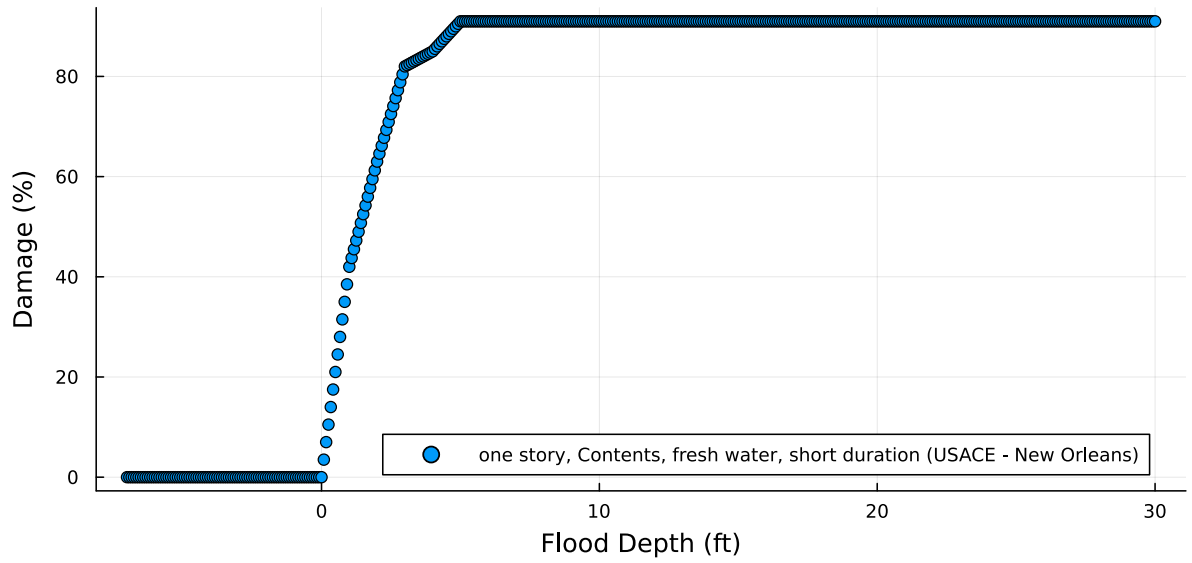
```
haz_fl_dept = CSV.read("data/haz_fl_dept.csv", DataFrame) # read in the file
desc = "one story, Contents, fresh water, short duration"
row = @rsubset(haz_fl_dept, :Description == desc)[1, :] # select the row I want
dd = DepthDamageData(row) # extract the depth-damage data
damage_fn = get_depth_damage_function(dd.depths, dd.damages) # get the depth-damage function
```

```
#13 (generic function with 1 method)
```

We can plot this as before

```
p = let
    depths = uconvert.(u"ft", (-7.0u"ft"):(1.0u"inch"):(30.0u"ft"))
    damages = damage_fn.(depths)
    scatter(
        depths,
        damages;
        xlabel="Flood Depth",
```

```
        ylabel="Damage (%)",
        label="$(dd.description) ($(dd.source))",
        legend=:bottomright,
        size=(800, 400),
        linewidth=2,
    )
end
p
```




### 1.3 Annual expected flood damages

As discussed above, today we'll focus on calculating the annual expected cost of flooding. In the previous lab, we used a Monte Carlo approach to estimate the expected cost of flooding. We'll repeat that today. Recall that your offset will be different!

```
gauge_dist = GeneralizedExtremeValue(5, 1, 0.1) # hypothetical gauge distribution
offset = 7.5 # hypothetical height from house to gauge
house_dist = GeneralizedExtremeValue(gauge_dist.  - offset, gauge_dist. , gauge_dist. )

samples = rand(house_dist, 100_000) .* 1u"ft"                                    ①
damages = damage_fn.(samples)                                                    ②
expected_damages_pct = mean(damages)                                             ③
```

① Draw 100,000 samples from the distribution of flood heights at the house and add units of feet.
② Calculate the damages for each sample using our function.
③ Calculate the expected damages as the mean of the damages. This is the Monte Carlo strategy $\int p(x)f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x)$ which requires $x_i \sim p(x)$.

```
4.127262623117908
```

> **Tip**
>
> There's no magic reason why we need to use 100,000 samples. Although this runs extremely fast, you could use fewer samples if you wanted to. A good way to check that you have enough samples is to re-run the experiment a few different times, and then to make sure that your expected damages don't change much from run to run. Even with 100,000 samples, I see a change of about 0.2% from run to run.

The damages we have calculated are expressed as a percentage of the value of the house (structure and contents, not land). To convert this to a dollar value, we need to know the value of the house. This is of course tricky to estimate, but let's use an example value. For your analysis, use Zillow or Redfin or similar to get a sense of the value of a house in the area you're considering. Make some assumption about the fraction of the value that corresponds to the house structure relative to the land.

```
house_structure_value = 250_000
expected_damages_usd = house_structure_value * expected_damages_pct / 100
```

```
10318.156557794771
```

We can treat this as the expected cost of flooding for this model.

### 1.4 Cost of elevating

Next, we have the cost of elevating. We'll use equations I've used before Doss-Gollin & Keller (2023). Essentially, we have a piecewise linear function that depends on the area of the house and how height we elevate.

```
house_area = 1000u"ft^2"
```

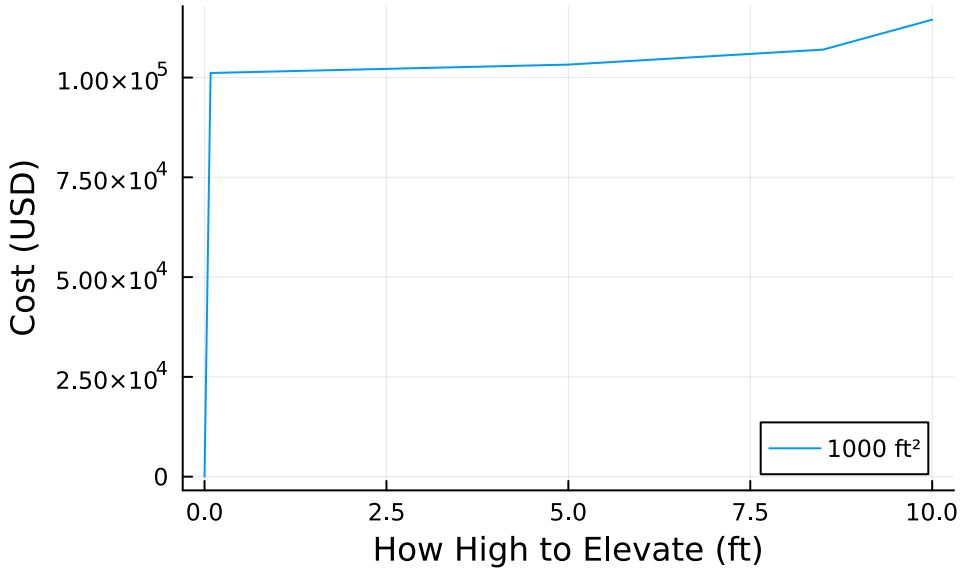
To get the cost function, we use the `get_elevation_cost_function()` function, which is defined in the `depthdamage.jl` file. This function fits an interpolator to the data, which we want because we don't want to have to re-fit the interpolator every time we want to calculate the cost of elevating the house.

```
elevation_cost = get_elevation_cost_function() # gives us a fitted interpolator
```

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

We can visualize this function as follows

```
heights = uconvert.(u"ft", (0u"ft"):(1u"inch"):(10u"ft")) # some heights we will consider
plot(
    heights,
    elevation_cost.(heights, house_area);
    xlabel="How High to Elevate",
    ylabel="Cost (USD)",
    label="$(house_area)",
    tiitle="Cost of Elevating a House",
)
```




Of course, this simple approach is masking lots of important characteristics of each house and region that affect how expensive it might be to elevate.

### 1.5 NPV analysis

We can use the functions above to calculate the NPV of elevating the house for a single year. However, if we are evaluating the house over a $T$ year design window (recall: cost-benefit analysis needs a specific time horizon over which costs and benefits are computed) then we need to discount the costs and benefits to the present. We do this as

$$\text{NPV} = \sum_{i=1}^{T} u_t(a, \mathbf{s})(1-r)^{i-1}$$

Let's say we have a 10 year design window and a discount rate of 5%. Let's say we elevate zero feet. In that case, the cost of elevating is zero, and the expected cost of flooding is `expected_damages_usd` every year (neglecting any sea-level rise). Then we can calculate the NPV as follows.

Then we can calculate the NPV as follows:

```
annual_damages = [expected_damages_usd for _ in 1:10] # annual expected damages
discount_rate = 0.05
npv = sum(annual_damages .* (1 - discount_rate) .^ (0:9))
```

```
82805.90163596644
```

another, more concise, way to write this is

```
npv2 = sum([expected_damages_usd * (1 - discount_rate)^(i - 1) for i in 1:10])
```

```
82805.90163596644
```

both are equivalent; you can use the one you prefer.

# 2 Your turn

Now it's your turn to do some analysis.

## 2.1 Single Year Function

First, we're going to write a function that tells us our costs and benefits in a single year. The information we'll need for that year is:

- The distribution of of flooding at the house
- The depth-damage function (as in percentage terms)
- The cost of elevating the house (and the house area)
- The house value
- How high we elevated the house **in that year**.

This will look something like this

```
function single_year_cost_benefit(flood_dist, damage_fn, elevation_cost, house_area, house_valu

    # calculate the expected damages
    c_dmg = ...

    # calculate the cost of elevating
    c_constr = ...

    # return the total cost and benefit
    return -c_constr - c_dmg
end
```

## 2.2 NPV Function

Next, we need to write a function that calculates the NPV over a $T$ year design window. This function will take in all the information needed for the `single_year_cost_benefit` function, as well as the number of years `T` and the discount rate. Then, it will call the `single_year_cost_benefit` function for each year, and discount the costs and benefits to the present. Be sure to set $\Delta h$ to zero feet (you'll get an error without units) for every year after the first!

```
function npv_cost_benefit(flood_dist, damage_fn, elevation_cost, house_area, house_value, Δh, T
    # calculate the costs and benefits for each year, and then discount
    # see above!
    return npv
end
```

## 2.3 One SOW, several actions

First, let's calculate the NPV for a single state of the world and two actions. Now that you have the `npv_cost_benefit` function, this should be straightforward. Guess how high you might want to elevate the house, and then calculate the NPV for that action.

Compare your elevation to zero feet, and explore a few other elevations. What do you notice?

### 2.4 Sensitivity test

Now let's perform a simple sensitivity test. Let's assume that the discount rate is uncertain, as we explored in class. Use a Monte Carlo approach to estimate the expected NPV for a range of discount rates. As an example, you could use `Normal(4, 2)`, which has a mean of 4 and a standard deviation of 2.

### 2.5 Discussion

1. What do you notice about the NPV for different actions?
2. What do you notice about the sensitivity test?
3. What are some limitations of this analysis?
    - What things are missing from this analysis that you think are important?
    - How might they affect the results?
    - What are some ways you might address these limitations?

Doss-Gollin, J., & Keller, K. (2023). A subjective Bayesian framework for synthesizing deep uncertainties in climate risk management. *Earth's Future*, *11*(1). https://doi.org/10.1029/2022EF003044

Zarekarizi, M., Srikrishnan, V., & Keller, K. (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