Lab 7: Parking Garage Case Study

CEVE 421/521

Fri., Mar. 8

1 Introduction

Neufville et al. (2006) introduced a case study of a parking garage in which the decision variable is the number of levels to build. This is about as simple as a sequential decision problem can get, which makes it a great "toy problem" to illustrate the basic concepts of sequential decision making and how to program them effectively.

1.1 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.

1.2 Load packages

```
using Revise
using ParkingGarage
```

and also regular packages

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

1.3 Formal problem framing

We view the problem as a sequential decision problem following Neufville et al. (2006). We have a single decision to make: how many levels to build. We will compare results for two cases:

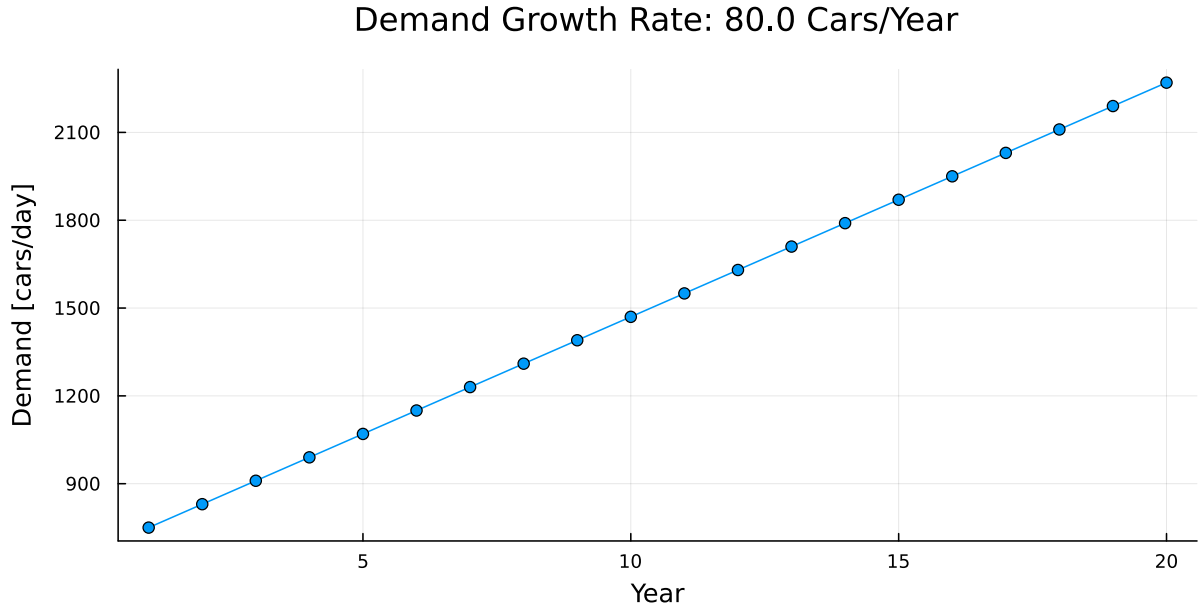
1. static case. The number of levels is fixed.

2. adaptive case. We pay an extra 5% for up-front costs, but then retain the option to build more levels in the future. We will use a simple rule to decide when to build more levels: if demand exceeds the current capacity, we will build one more level.

As we've seen in class, a key concept in sequential decision making is the idea of a state. In this problem, we have three state variables: the year and the number of levels. We could add some complexity to our problem by making the demand stochastic, in which case we'd want it to be a state variable, but here we'll treat it as a determinstic function of time.

We also have some uncertainty in our model: the discount rate, the time horizon, and the demand growth rate. The paper uses an exponential growth model for demand, but we'll use a linear one.

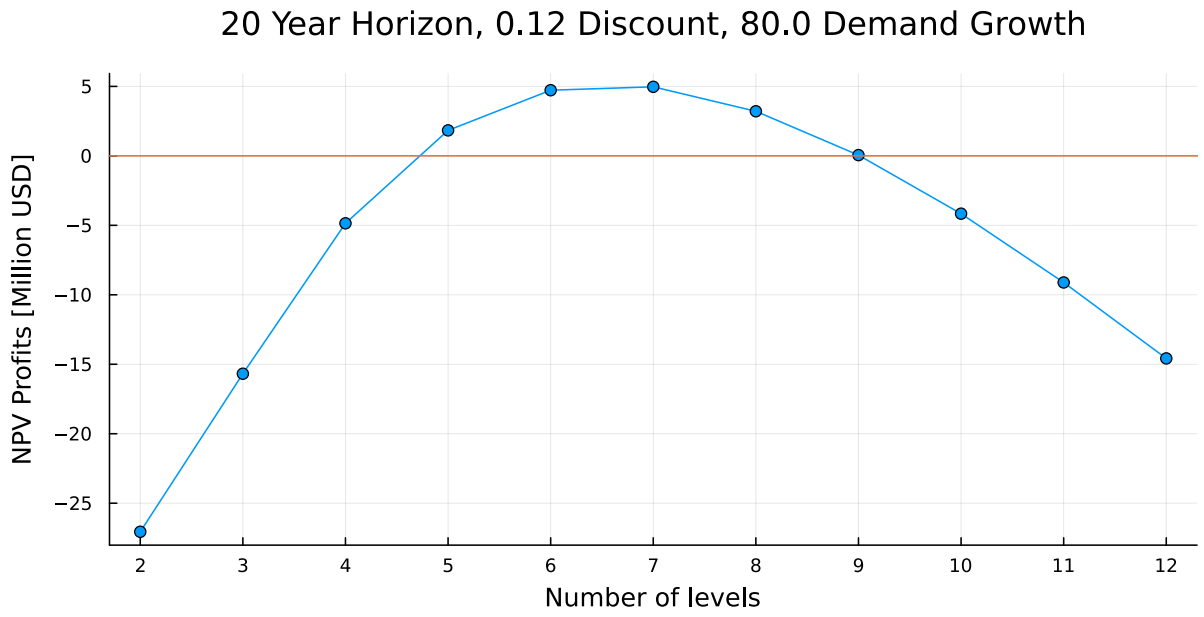
```
let
    sow = ParkingGarageSOW()
    years = 1:(sow.n_years)
    demand = [
        ParkingGarage.calculate_demand(year, sow.demand_growth_rate) for year in years
    ]
    plot(
        years,
        demand;
        ylabel="Demand [cars/day]",
        xlabel="Year",
        legend=false,
        title="Demand Growth Rate: $(sow.demand_growth_rate) Cars/Year",
        size=(800, 400),
        marker=:circle,
    )
end
```



2 Static case

This function assumes that the demand is deterministic and that the number of levels is fixed. The decision variable is the number of levels of the garage to build. If we consider a single SOW, we can calculate the NPV of the profits for a given policy.

```
let
    sow = ParkingGarageSOW(; demand_growth_rate=80.0, n_years=20, discount_rate=0.12)
    n_levels = 2:12
    policies = [StaticPolicy(i) for i in n_levels]
    profits = [simulate(sow, policy) for policy in policies]
    plot(
        n_levels,
        profits;
        ylabel="NPV Profits [Million USD]",
        xlabel="Number of levels",
        legend=false,
        title="$(sow.n_years) Year Horizon, $(sow.discount_rate) Discount, $(sow.demand_growth_
        size=(800, 400),
        marker=:circle,
        xticks=n_levels,
    )
    hline!([0])
end
```



2.1 Uncertainty

Figure 1 of Neufville et al. (2006) shows how the NPV changes when uncertainty is added to the model. Reproduce this figure, using our model. Specifically:

1. Generate an ensemble of SOWs. Justify how you are sampling the three parameters (`n_years`,

`demand_growth_rate`, and `discount_rate`). I suggest to keep `n_years` as a constant, and perhaps to keep the discount rate constant as well.
2. For each SOW, calculate the NPV for each policy.
3. Calculate the average NPV for each number of levels and plot.

3 Adaptive case

The static case sheds some light on decision making under uncertainty. However, the point of the (**denuefville_parkinggarage:2006?**) paper is to illustrate the value of flexibility in decision making.

To implement this, you'll need to get your hands a bit dirty with the source code. Specifically, you need to edit the `function get_action(x::ParkingGarageState, policy::AdaptivePolicy)` function in `ParkingGarage/src/sim.jl`. You'll need to use `if...else...end` statements to implement the adaptive policy. We'll talk about this in class!

Once you've implemented this function, you can simulate the adaptive policy and compare the NPV to the static policy. Compare the fixed and adaptive policies for both the deterministic (single SOW) and stochastic (ensemble of SOWs) cases. Plot the NPV as a function of the number of levels for each case.

Neufville, R. de, Scholtes, S., & Wang, T. (2006). Real Options by Spreadsheet: Parking Garage Case Example. *Journal of Infrastructure Systems*, *12*(2), 107–111. https://doi.org/10.1061/(asce)1076-0342(2006)12:2(107)