Day 3: Supply II

We talked today about environmental regulation in electricity markets.

We will learn today how to build a model of an electricity market with incomplete regulation ("leakage").

The data and code are based on the paper "Border Carbon Adjustments When Carbon Intensity Varies Across Producers: Evidence from California," by Meredith Fowlie, Claire Petersen, and Mar Reguant.

We first load relevant libraries.

```
begin
    using CSV    , Tables
    using JuMP      # used for mathematical programming
    using Ipopt      # solver
    using Printf
    using Plots
    using StatsPlots   , Statistics   , StatsBase
end
```

```
# A trick to make the notebook wider
# This is not saying great things about the length of my coding lines, including this line.
html"""<style>
main {
    max-width: 900px;
}
"""
```

Data

The data have been generated by compiling several publicly-available sources about the market outcomes of the Western interconnection areas. This is a recent dataset (2019 data) benefiting from better availability of data on wind and solar production available at EIA.

Most of the data are region-hourly, and data on power plants is plant-region.

Here I provide a description of the variables (available in the online appendix as well).

- a: Electricity demanded when price is zero.
- b: Slope of electricity demand curve.
- qmr: Quantity of electricity from wind, solar, nuclear, and hydroelectric energy sources, which “must run” in the model.
- mc: Marginal cost of electricity production.
- er: Emissions rate.
- mw: Capacity.
- cf: Capacity factor.
- flag: Dummy variable flagging natural gas units near large cities to mandate a minimum required generation.
- isca: Dummy variable indicating the generating unit is in California.
- istax: Dummy variable indicating the generating unit is in California’s tax jurisdiction.
- lines: Maximum capacity that can flow through transmission lines.
- fct: Distribution of electricity flow originating in a region along the transmission lines.

The files are stored as .CSV matrices, ready to be imported and used in the model. These CSV files were generated by preparation code from raw sources and are subject to assumptions, e.g., for the marginal cost of the power plants and clustering techniques to simplify the regions and the number of observations.

```
begin
    # we load a bunch of matrices that were created via other datasets
    datasets = ["a","b","cf","er","fct","flag","fuel",
        "isca","ischp","istax","lines",
        "mc","mw","qmr","qrw","w"];

    # we put them into data
    data = Dict{String,Any}();
    for x in datasets
        data[string(x)] = CSV.File(string("data_leakage/",x,".csv"), header=false) |> Tables.matrix;
    end
end
```

```
4×100 Matrix{Float64}:
 33.1375    30.005     36.6204  28.4611   …  32.6258    33.4986  40.5096    26.1449
 33.8523    29.3415    42.0462  31.6467      37.3512    41.5628  38.3515    26.8072
 12.2017    13.1445    14.2157  10.9905      11.3517    12.4546  19.2729    10.1063
  8.44305    7.52042   10.5337   8.11629      9.18321    9.768    9.21859    7.07419
```

```
data["a"]  # Example of output, intercept of demand for 4 regions and 100 periods
```

```
Dict("b" ⇒ 4×100 Matrix{Float64}:                                                                , "lines" ⇒ 5×1 Mat
            0.0900063  0.0900063  0.0900063  0.0900063  …  0.0900063  0.0900063  0.0900063            6.85
            0.0982788  0.0982788  0.0982788  0.0982788     0.0982788  0.0982788  0.0982788            7.13
            0.0444894  0.0444894  0.0444894  0.0444894     0.0444894  0.0444894  0.0444894            0.895
            0.0253585  0.0253585  0.0253585  0.0253585     0.0253585  0.0253585  0.0253585            1.1075
                                                                                                      1.145
```

```
data
```

Model

We will be solving the model using the "maximization" approach of optimizing economic surprlus. This will save us from coding up the transmission lines constraints using mixed integer programming.

The same "mazimization" approach is used in the paper "Strategic Policy Choice in State-Level Regulation: The EPA's Clean Power Plan." by Bushnell, James B., Stephen P. Holland, Jonathan E. Hughes, and Christopher R. Knittel. We build on this model, which is available in AMPL at **http://doi.org/10.3886/E114648V1**.

In comparison, one big limitation of our short conference paper is that it does not include investment.

The model follows similar steps from yesterday:

1. Declare the model to initialize it. We also call it "model" here, but it could be any name.
2. Declare some useful indices to keep track of regions, plants, and lines. [Note: There is no time here because the function will only solve one hour at a time]
3. [New] Declare some parameters as a function of the policy case, which is an input to the function.
4. Declare variables.
5. Declare objective function.
6. Declare constraints, [New] including those for the environmental regulation and the transmission lines.

We consider several cases.

1. No regulation, tax is 0.
2. Uniform tax, every region.
3. CA tax only.
4. Tax of imports at default rate, with opt-out.
5. Tax of imports at default rate, no opt-out.

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

```julia
function clear_market_at_t(d::Dict{String,Any};
        t=1, case=2, tax=17.0, default=0.428, beta=1.0)

    model = Model(
        optimizer_with_attributes(
            Ipopt.Optimizer, "print_level"=>0)
        );

    # Set market inputs
    R = size(d["a"],1);  # number of regions
    U = size(d["mc"],2);
    L = size(d["lines"],1);

    p0 = d["a"]./d["b"]

    # define costs of carbon depending on policy
    if case==1  # no regulation
        er_tax = d["er"];
        tax = 0.0;
    elseif case==2  # uniform tax
        er_tax = d["er"];
    elseif case==3  # only CA tax
        er_tax = d["er"] .* d["istax"];
    elseif case==4  # opt-out default
        er_tax = d["er"] .* d["istax"] .+ ((d["er"] .> default) .* default
            + (d["er"] .<= default) .* d["er"]) .* (1. .- d["istax"]);
    elseif case==5  # mandatory default
        er_tax = d["er"] .* d["istax"] .+ default .* (1. .- d["istax"]);
    else
        return @printf("Invalid case");
    end

    # variables to solve for
    @variable(model, price[1:R]);
    @variable(model, demand[1:R]);
    @variable(model, yflow[1:R-1]);  # swing node is CA
    @variable(model, q[1:R, 1:U]>=0);
    @variable(model, q_ca[1:R, 1:U]>=0);  # quantity sent to CA for accounting purposes
    @variable(model, qmr_ca[1:R]>=0);  # quantity must-run sent to CA (no emissions)

    # summary variables
    @variable(model, surplus);
    @variable(model, totalcost);  # fuel cost
    @variable(model, totalecost);  # taxed emissions costs
    @variable(model, totale[1:R]);
    @variable(model, totale_ca);  # emissions based on accounting quantity
    @variable(model, totale_ca_claimed);  # emissions based on accounting quantity but the true rate
    @variable(model, totale_ca_instate);  # emissions based on accounting quantity but the true rate

    @NLobjective(model, Max, surplus - totalcost - totalecost);

    # definition of objective function
    @constraint(model, surplus == sum(0.5 * (p0[r, t] + price[r]) * demand[r] for r in 1:R));
    @constraint(model, totalcost == sum(q[r,i] * d["mc"][r, i] for r in 1:R, i in 1: U));
    if case==2
        @constraint(model, totalecost == sum(q[r, i] * d["er"][r, i] * tax for r in 1:R, i in 1: U));
    elseif case==5
        @constraint(model, totalecost == sum(q_ca[r, i] * er_tax[r, i] * tax for r in 1:R, i in 1: U)
            + sum(qmr_ca[r] * default * tax for r in 2:R));
    else
        @constraint(model, totalecost == sum(q_ca[r, i] * er_tax[r, i] * tax for r in 1:R, i in 1: U));

    end
    @constraint(model, [r=1:R], totale[r] == sum(q[r, i] * d["er"][r, i] for i in 1: U));
    @constraint(model,  totale_ca == sum(q_ca[r, i] * d["er"][r, i] for r in 1:R, i in 1: U));
    @constraint(model, totale_ca_instate ==
        sum(q_ca[r, i] * d["er"][r, i] * d["isca"][r, i] for r in 1:R, i in 1: U));
    if case==5
        @constraint(model, totale_ca_claimed == sum(q_ca[r, i] * er_tax[r, i] for r in 1:R, i in 1: U)
            + sum(qmr_ca[r] * default for r in 2:R));
    else
        @constraint(model, totale_ca_claimed == sum(q_ca[r, i] * er_tax[r, i] for r in 1:R, i in 1: U));
    end

    # definition demand
    @constraint(model, demand[1] == (d["a"][1, t] - d["b"][1, t] * price[1]));
    @constraint(model, [r=2:R], demand[r] == (d["a"][r, t] - d["b"][r, t] * price[r]));

    # constraint quantities to 95% of name plate capacity
    @constraint(model, [r=1:R, i=1:U], q[r, i] <= 0.95 *  d["mw"][r, i]/1000.0);
```

```
    # congestion constraint
    @constraint(model, [r=1:R, i=1:U], q[r, i] >= d["cf"][r, i] * d["flag"][r, i] * d["mw"][r, i]/1000.0);

    # market clearing
    @constraint(model, demand[1] == sum(q[1,i] for i=1:U) + d["qmr"][1, t] + sum(yflow[z] for z in 1:R-1));
    @constraint(model, [r=2:R], demand[r] + yflow[r-1] == sum(q[r,i] for i=1:U) + d["qmr"][r, t]);

    # transmission line
    @constraint(model, [l=1:L], sum(d["fct"][z,l] * yflow[z] for z in 1:R-1) <= d["lines"][l]);
    @constraint(model, [l=1:L], sum(d["fct"][z,l] * yflow[z] for z in 1:R-1) >= -d["lines"][l]);

    # california accounting
    @constraint(model, demand[1] == sum(q_ca[r,i] for r in 1:R, i in 1: U) + sum(qmr_ca[r] for r in 1:R));
    @constraint(model, [r=1:R, i=1:U], q_ca[r, i] <=  q[r, i]);  # CA quantity
    @constraint(model, [r=1:R, i=1:U], q_ca[r, i] >=  d["istax"][r,i] * q[r, i]);  # equal to q if CA/taxed
    @constraint(model, [r=1:R, i=1:U], q_ca[r, i] >=  d["isca"][r,i] * q[r, i]);  # equal to q if CA/taxed
    @constraint(model, qmr_ca[1] == d["qmr"][1, t]);  # equal to q if CA
    @constraint(model, [r=2:R], qmr_ca[r] <=  beta * d["qmr"][r, t]);  # CA quantity from hydro + renewables

    optimize!(model)

    status = @sprintf("%s",JuMP.termination_status(model));

    if ((status=="LOCALLY_SOLVED") | (status=="ALMOST_LOCALLY_SOLVED"))
        results = Dict("status" => status,
                "surplus" => JuMP.value.(surplus),
                "totalcost" => JuMP.value.(totalcost),
                "totalecost" => JuMP.value.(totalecost),
                "totale" => JuMP.value.(totale),
                "totale_ca" => JuMP.value.(totale_ca),
                "totale_ca_claimed" => JuMP.value.(totale_ca_claimed),
                "totale_ca_instate" => JuMP.value.(totale_ca_instate),
                "price" => JuMP.value.(price),
                "demand" => JuMP.value.(demand),
                "yflow" => JuMP.value.(yflow),
                "q" => JuMP.value.(q),
                "q_ca" => JuMP.value.(q_ca),
                "qmr_ca" => JuMP.value.(qmr_ca));
        return results
    else
        results = Dict("status" => status);
        return results
    end

end
```

```
Dict("price" ⇒ [38.0763, 42.9465, 33.2061, 40.0697], "totale_ca_claimed" ⇒ 6.98739, "totale_ca" ⇒ 6.98739, "s
```

```
clear_market_at_t(data, tax=17.0, case=2)
```

Now that we have a function that generates outcomes from one hour, we can loop to get results for the entire period.

Disclaimer: This is not the cleanest way of doing this. I have been lately using DataFrames for cleaner handling. But this has the advantage of preserving a lot of detail about the solution.

Warning: This loop can take a bit of time. Feel free to reduce the number of periods it is looping over by modifying the time loop.

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

```julia
function clear_market_loop(d::Dict{String,Any}; T=100, case=2, tax=17.0, default=0.428, beta=1.0)

    # prepare buckets
    status = Any[];
    surplus = Any[];
    totalcost = Any[];
    totalecost = Any[];
    totale = Any[];
    totale_ca = Any[];
    totale_ca_claimed = Any[];
    totale_ca_instate = Any[];
    price = Any[];
    demand = Any[];
    yflow = Any[];
    q = Any[];
    q_ca = Any[];
    qmr_ca = Any[];
    w_mat = Any[];

    T = min(T, size(d["a"],2));

    for t in 1:T # limit to make computation faster!
        res = clear_market_at_t(data, t=t,
            case=case, tax=tax, default=default, beta=beta)
        if ((res["status"]=="LOCALLY_SOLVED") | (res["status"]=="ALMOST_LOCALLY_SOLVED"))
            push!(status, res["status"]);
            push!(surplus, res["surplus"]);
            push!(surplus, res["surplus"]);
            push!(totalcost, res["totalcost"]);
            push!(totalecost, res["totalecost"]);
            push!(totale, res["totale"]);
            push!(totale_ca, res["totale_ca"]);
            push!(totale_ca_claimed, res["totale_ca_claimed"]);
            push!(totale_ca_instate, res["totale_ca_instate"]);
            push!(price, res["price"]);
            push!(demand, res["demand"]);
            push!(yflow, res["yflow"]);
            push!(q, res["q"]);
            push!(q_ca, res["q_ca"]);
            push!(qmr_ca, res["qmr_ca"]);
            push!(w_mat, d["w"][t]);
        else
            return @sprintf("Hour %d failed with status %s!", t, res["status"]);
        end

    end

    results = Dict("status" => status,
                "surplus" => surplus,
                "totalcost" => totalcost,
                "totalecost" => totalecost,
                "totale" => totale,
                "totale_ca" => totale_ca,
                "totale_ca_claimed" => totale_ca_claimed,
                "totale_ca_instate" => totale_ca_instate,
                "price" => price,
                "demand" => demand,
                "yflow" => yflow,
                "q" => q,
                "q_ca" => q_ca,
                "qmr_ca" => qmr_ca,
                "w" => w_mat);

    return results

end
```

```julia
Tlimit = 10;  # We put a limit on the number of hours to run as it is intensive
```

```julia
case2 = clear_market_loop(data, tax=17.0, case=2, T=Tlimit);
```

A note on the structure of the data as these things can be very **language-specific**.

The dictionary stores the matrix for each variable and day into the dictionary entry with its variable name. Each day is a separate matrix/vector in itself. So to access day t, we use:

```
full_results["price"][t]
```

which is a vector of regional prices.

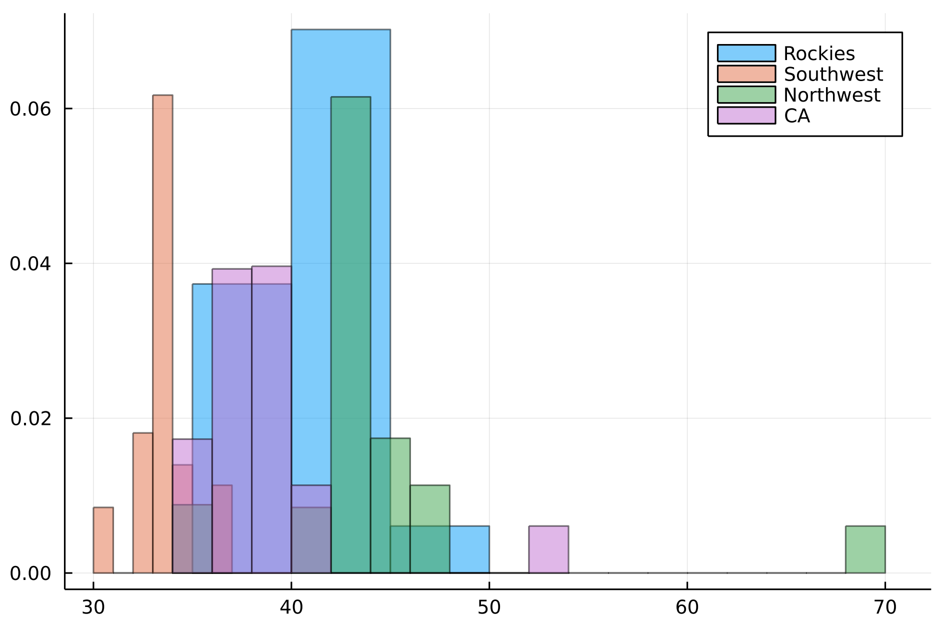
```
[[38.0763, 42.9465, 33.2061, 40.0697], [37.7952, 42.5413, 33.0491, 38.4369], [39.0705, 44.8298, 33.3112, 44.1077
```

```
case2["price"]
```

Examining policy outcomes

We can explore the results of the policy for alternative combinations of parameters.

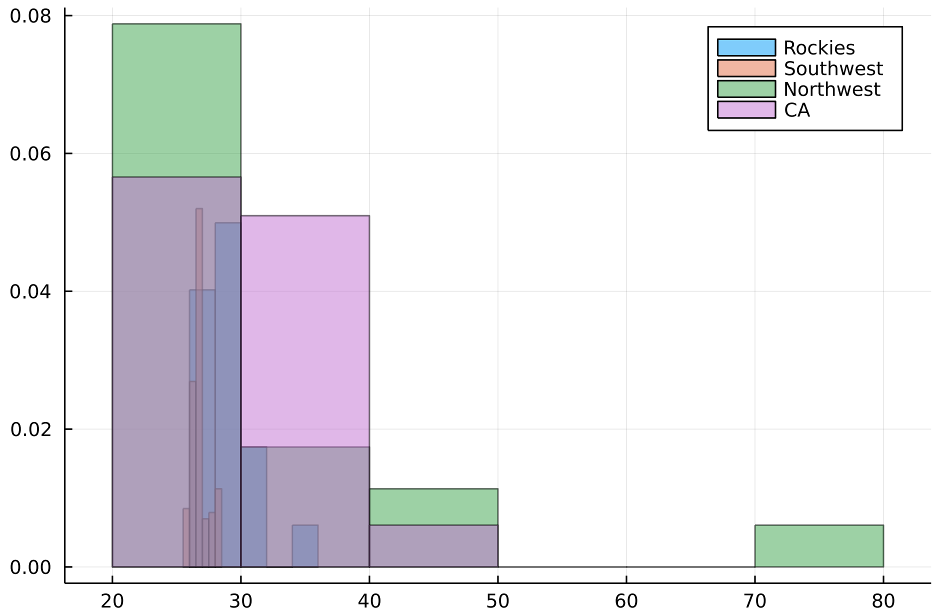
First, we examine the baseline uniform case with a tax of 17/tCO2e and its implications for prices.




```
let

    T = size(case2["w"], 1);

    p1 = [case2["price"][t][1] for t=1:T];
    p2 = [case2["price"][t][2] for t=1:T];
    p3 = [case2["price"][t][3] for t=1:T];
    p4 = [case2["price"][t][4] for t=1:T];
    w = [case2["w"][t] for t=1:T];

    histogram(p4, weights=w, label="Rockies", alpha=.5)
    histogram!(p3, weights=w, label="Southwest", alpha=.5)
    histogram!(p2, weights=w, label="Northwest", alpha=.5)
    histogram!(p1, weights=w, label="CA", alpha=.5)

end
```

Let's compare this to a case in which only California is taxed (case 3), leading to much lower prices particularly for NW and Rockies but all markets in general due to links across markets.

```
case3 = clear_market_loop(data, tax=17.0, case=3, T=Tlimit);
```




```
let

    T = size(case3["w"], 1);

    p1 = [case3["price"][t][1] for t=1:T];
    p2 = [case3["price"][t][2] for t=1:T];
    p3 = [case3["price"][t][3] for t=1:T];
    p4 = [case3["price"][t][4] for t=1:T];
    w = [case3["w"][t] for t=1:T];

    histogram(p4, weights=w, label="Rockies", alpha=.5)
    histogram!(p3, weights=w, label="Southwest", alpha=.5)
    histogram!(p2, weights=w, label="Northwest", alpha=.5)
    histogram!(p1, weights=w, label="CA", alpha=.5)

end
```

Emissions go away from California in this new setting.

We can compare emissions in California between case2 and case3.

59.183110796107066

```
sum([case2["totale_ca_instate"][t] for t=1:size(case2["w"],1)])
```

36.293768889999654

```
sum([case3["totale_ca_instate"][t] for t=1:size(case3["w"],1)])
```

We can analyze the impact of default policies into the outcome, which is absent unless firms cannot reshuffle.

```
case4 = clear_market_loop(data, tax=17.0, case=4, T=Tlimit);
```

36.29376892212316

```
sum([case4["totale_ca_instate"][t] for t=1:size(case4["w"],1)])
```

```
case5 = clear_market_loop(data, tax=17.0, case=5, T=Tlimit);
```

43.84908331284188

```
sum([case5["totale_ca_instate"][t] for t=1:size(case5["w"],1)])
```

Follow-up exercises

1. Modify the output code in the market loop so that you can make it a choice in the function: detailed (debugging) output, hourly output, or summary output.
2. Try different default rates for case 5. You will notice that emissions start declining as default rates start increasing, replicating the main figure in the paper.
3. What are the tensions between balancing leakage and efficiency? Try to show a more comprehensive measure of welfare than what our social planner is using (which only accounts for taxed emissions).