Day 6: Supply II - Transmission and Leakage

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 and set the path.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from pyomo.environ import *
```

✓ 1.1s Python

```
dirpath = "/Users/marreguant/Dropbox/TEACHING/BSE/Electricity2026/day6/practicum/"
```

✓ 0.0s Python

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 for each region
- 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.
- w: Weight.

This will be a matrix for each input

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.

```
# 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[str, np.ndarray] = {}

for x in datasets:
    fp = f"{dirpath}/data_leakage/{x}.csv"
    # Tables.matrix -> plain numeric matrix -> NumPy array
    data[x] = pd.read_csv(fp, header=None).to_numpy()
```

[6] ✓ 0.0s Python

I am putting all the data in one big “bucket” (Dict)

```
data["fct"]  # intercept of demand for 4 regions and 100 periods
```

[7] ✓ 0.0s Python

```
array([[ 0.623,  0.378, -0.144,  0.144,  0.234],
       [ 0.378,  0.623,  0.144, -0.144, -0.234],
       [ 0.5  ,  0.5  ,  0.5  ,  0.5  ,  0.   ]])
```

With that, every unit of electricity produced in each of the 4 regions will flow across regions according to flow factors `fct` (taking CA as reference)

```
# The amount of transmission available between states
data["lines"]
```

2] ✓ 0.0s Python

```
array([[6.85  ],
       [7.13  ],
       [0.895 ],
       [1.1075],
       [1.145 ]])
```

```
# How power from each of the other regions gets to California (which path it takes)
data["fct"]
```

3] ✓ 0.0s Python

```
array([[ 0.623,  0.378, -0.144,  0.144,  0.234],
       [ 0.378,  0.623,  0.144, -0.144, -0.234],
       [ 0.5  ,  0.5  ,  0.5  ,  0.5  ,  0.   ]])
```

This matches the grid we discussed.
In practice, these factors and lines can have many entries.

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 "maximization" 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 last week:

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. Declare variables.
4. Declare objective function.
5. Declare constraints, [New] including the transmission lines.

We will start by considering the case where all electricity producers are regulated under the same carbon pricing regime.

```
def clear_market_at_t_simple(d: dict, t: int = 1, tax: float = 17.0):
    """
    Pyomo translation of the JuMP/Ipopt model for leakage
    """

    # --- dimensions ---
    R = d["a"].shape[0]          # number of regions
    U = d["mc"].shape[1]         # number of plants
    L = d["lines"].shape[0]      # number of lines

    # p0 = a ./ b (intercept of inverse demand)
    p0 = d["a"] / d["b"]

    # --- model ---
    m = ConcreteModel()

    # Sets
    m.R = RangeSet(0, R - 1)
    m.Rother = RangeSet(1, R - 1)
    m.U = RangeSet(0, U - 1)
    m.Z = RangeSet(0, R - 2)   # yflow
    m.L = RangeSet(0, L - 1)

    # Decision variables
    m.price = Var(m.R)                  # unrestricted
    m.demand = Var(m.R)                 # unrestricted
    m.yflow = Var(m.Z)                  # unrestricted
    m.q = Var(m.R, m.U, domain=NonNegativeReals)
```

Notice we do not have time, as this will solve only one hour!

Demand and prices are now by regions, and we need to track flows

```
# Summary variables
m.surplus = Var()
m.totalcost = Var()
m.totalecost = Var()
m.totale = Var(m.R)

# Objective: Max surplus - totalcost - totalecost
m.obj = Objective(expr=m.surplus - m.totalcost - m.totalecost, sense=maximize)

# --- constraints ---
# Surplus
m.surplus_def = Constraint(
    expr=m.surplus
    == sum(0.5 * (p0[r, t] + m.price[r]) * m.demand[r] for r in m.R)
)

# Fuel cost
m.totalcost_def = Constraint(
    expr=m.totalcost
    == sum(m.q[r, i] * d["mc"][r, i] for r in m.R for i in m.U)
)

# Emissions tax cost (all of them here)
m.totalecost_def = Constraint(
    expr=m.totalecost
    == sum(m.q[r, i] * d["er"][r, i] * tax for r in m.R for i in m.U)
)
```

The planner maximizes surplus taking into account emissions (first best)

```python
# Demand system
m.demand_def = Constraint(m.R,
    rule=lambda m, r: m.demand[r] == d["a"][r, t] - d["b"][r, t] * m.price[r],
)

# Capacity upper bound
m.cap_ub = Constraint(m.R,m.U,
    rule=lambda m, r, i: m.q[r, i] <= 0.95 * d["mw"][r, i] / 1000.0,
)

# Minimum generation
m.min_gen = Constraint(m.R, m.U,
    rule=lambda m, r, i: m.q[r, i] >= d["cf"][r, i] * d["flag"][r, i] * d["mw"][r, i] / 1000.0,
)
# Market clearing:
# CA = region 0 (swing node)
m.mkt_clear_CA = Constraint(
    expr=(
        m.demand[0]
        == sum(m.q[0, i] for i in m.U)
        + d["qmr"][0, t]
        + sum(m.yflow[z] for z in m.Z)
    )
)
# Other regions
m.mkt_clear_other = Constraint(m.Rother, rule= lambda m, r:
    m.demand[r] + m.yflow[r - 1] == sum(m.q[r, i] for i in m.U) + d["qmr"][r, t])

# Transmission constraints for each line l:
# -lines[l] <= sum(fct[z,l] * yflow[z]) <= lines[l]
m.line_ub = Constraint(m.L, rule=lambda m, l:
        sum(d["fct"][z, l] * m.yflow[z] for z in m.Z) <= d["lines"][l])
m.line_lb = Constraint(m.L, rule=lambda m, l:
        sum(d["fct"][z, l] * m.yflow[z] for z in m.Z) >= -d["lines"][l])
```

0s Python

Supply = Demand takes into account the flows.
Note: California receives flows as production, for others, this is defined as “extra demand” that they need to produce.

Adding transmission is simply two lines that limit flows!

```
res = solver.solve(m, tee=False)

# Map solver status to something close to JuMP's strings
ok = (res.solver.termination_condition in (
    TerminationCondition.optimal,
    TerminationCondition.locallyOptimal,
))

status = str(res.solver.termination_condition)

if ok:
    # Helper to extract values into numpy arrays matching Julia shapes
    price = np.array([value(m.price[r]) for r in m.R])
    demand = np.array([value(m.demand[r]) for r in m.R])
    yflow = np.array([value(m.yflow[z]) for z in m.Z])  # length R-1

    q = np.empty((R, U), dtype=float)
    for r in m.R:
        for i in m.U:
            q[r - 1, i - 1] = value(m.q[r, i])

    totale = np.array([value(m.totale[r]) for r in m.R])

    return {
        "status": status,
        "surplus": value(m.surplus),
        "totalcost": value(m.totalcost),
        "totalecost": value(m.totalecost),
        "totale": totale,
        "price": price,
        "demand": demand,
        "yflow": yflow,
        "q": q,
    }
```

0s

As usual, we output relevant outcomes

```
results_t= clear_market_at_t_simple(data, tax=20.0)
```

✓ 0.2s

```
print(results_t["price"])
```

✓ 0.0s

```
[39.33275489 44.77014451 33.89536527 39.83019528]
```

We now have regional prices

Creating a function that will run the market for many days

We create a `clear_market_loop` function that will run several days and store the results.

```python
def clear_market_loop(d: dict, T: int = 100, case: int = 2, tax: float = 17.0, default: float = 0.428):
    """
    This will solve the model for several time periods.
    """

    # prepare buckets
    out = {
        "status": [],
        "surplus": [],
        "totalcost": [],
        "totalecost": [],
        "totale": [],
        "price": [],
        "demand": [],
        "yflow": [],
        "q": [],
        "w": [],
    }

    T_eff = min(T, d["a"].shape[1])
```

We store the equilibrium variables for each time period that we solve

```
for t in range(T_eff):
    res = clear_market_at_t_simple(d, t=t)

    if res.get("status") in (TerminationCondition.optimal,TerminationCondition.locallyOptimal):
        out["status"].append(res["status"])
        out["surplus"].append(res["surplus"])
        out["totalcost"].append(res["totalcost"])
        out["totalecost"].append(res["totalecost"])
        out["totale"].append(res["totale"])
        out["price"].append(res["price"])
        out["demand"].append(res["demand"])
        out["yflow"].append(res["yflow"])
        out["q"].append(res["q"])
        out["w"].append(d["w"][t] if isinstance(d["w"], np.ndarray) else d["w"][t])
    else:
        # Julia returns a formatted string and exits early
        return f"Hour {t} failed with status {res.get('status')}!"

return out
```

This code loops over many hours, each separately. It could be useful depending on your project.

A note on the structure of the data as these things can be very **specific** (different shapes for each object).

We can convert the variables to data frames so that we can handle them more easily.

```
# Example to get a price dataset
df_price = pd.DataFrame(results["price"])
df_price = df_price.rename(columns={0: "price_CA", 1: "price_NW", 2: "price_SW", 3: "price_RK"})
df_price["weights"] = np.array(results["w"])
```

✓ 0.0s

```
df_price
```

✓ 0.0s

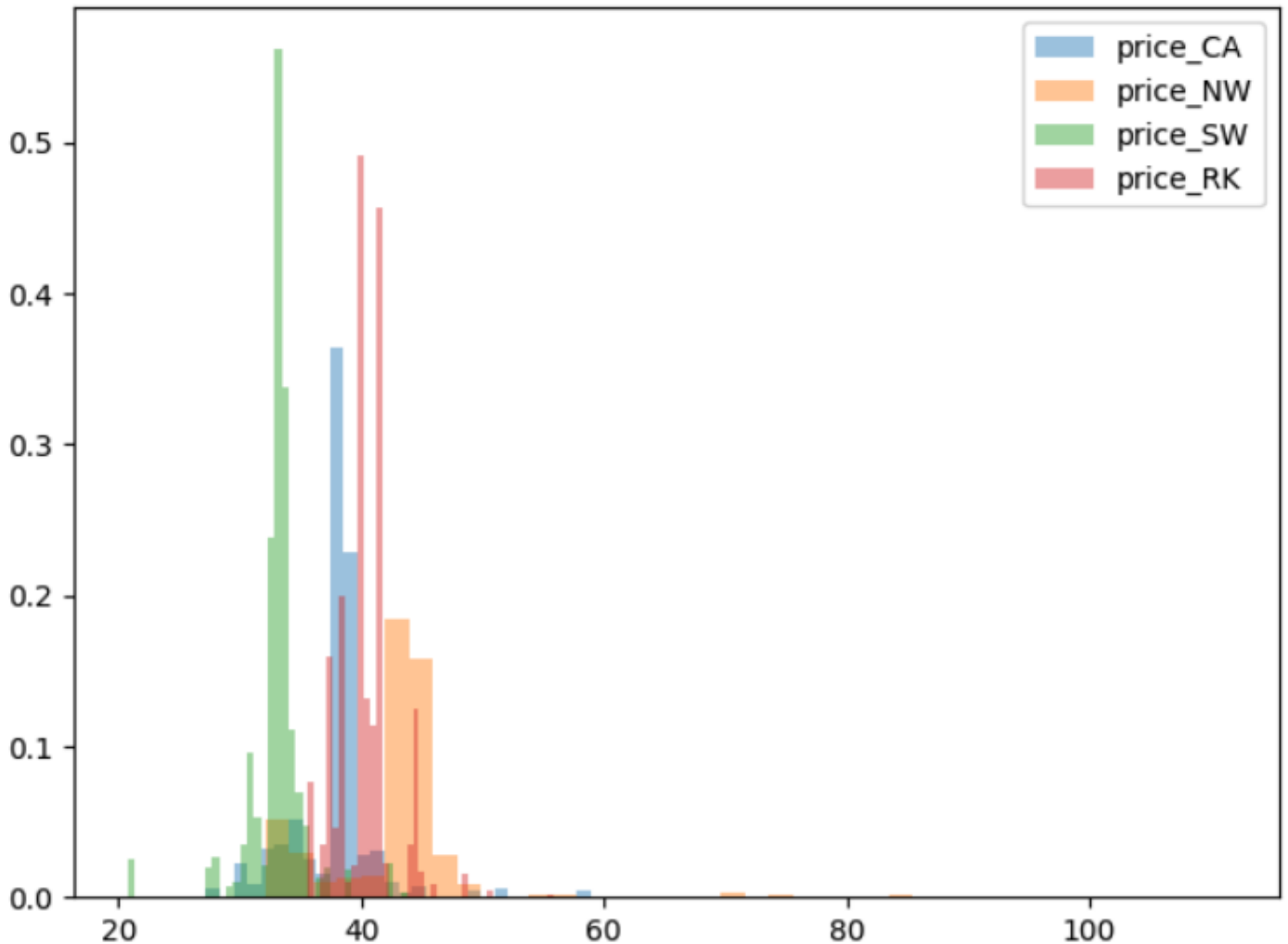
	price_CA	price_NW	price_SW	price_RK	weights
0	38.076269	42.946488	33.206050	40.069689	0.006987
1	37.795176	42.541251	33.049102	38.436876	0.013286
2	39.070479	44.829802	33.311156	44.107720	0.017409
3	35.922071	41.204206	30.639936	40.069689	0.008476

```python
# histogram of the four regions in the price dataframe
plt.figure()
for col in ["price_CA", "price_NW", "price_SW", "price_RK"]:
    plt.hist(df_price[col],
             weights=df_price["weights"],
             bins=40, alpha=0.4, density=True,
             label=str(col))

plt.legend()
plt.tight_layout()
plt.show()
```

✓ 0.1s

We can see transmission being very binding. How do patterns and flows change with different carbon policies?



Follow-up exercises

1. Modify the output code in the market loop so that it is just one row per time period, making it easier to save as a dataframe directly.
2. Try different taxes on emissions to see how they impact the results. Also try to compare taxing only California to taxing all states.
3. Could you do a Hausman-style (2025) calculation? What are the benefits of removing all transmission constraints according to this model?