Day 9: Demand - Solar rooftop

In this lecture, we talked about net metering policies and how they impact the profitability of solar panels along income groups.

We will consider the general equilibrium effects of solar rooftop using our model.

Note 1: This will be a limited simulation with only one representative consumer. To make this more interesting, we should have different types of consumers and income groups.

Note 2: Allowing for different types of consumers also shows good and bad selection: under good NEM policies, the "good" type of consumers who consume when solar is coincident are encouraged to enter.

We load the libraries and set the path.

```
using DataFrames
using CSV
using JuMP
using Ipopt, Cbc, HiGHS
using Plots
using Printf
using StatsBase
```

[1] ✓ 10.7s

```
dirpath = "/Users/marreguant/Dropbox/TEACHING/BSE/Electricity2025/day9/practicum/"
```

[24] ✓ 0.0s

```
"/Users/marreguant/Dropbox/TEACHING/BSE/Electricity2025/day9/practicum/"
```

Building the model

We load the same data as usual, and also clean it up to simplify it further and create the demand and import curves.

```
dfclust = CSV.read(string(dirpath,"data_jaere_clustered.csv"), DataFrame);

# Re-scaling (we multiply by 8.76 to make it into a full year of hours (divided by 1000))
dfclust.weights = 8.76 * dfclust.weights / sum(dfclust.weights);

# Here only one demand type to make it easier
dfclust.demand = dfclust.q_residential + dfclust.q_commercial + dfclust.q_industrial;

# Calibrate imports (using elas 0.3)
dfclust.bm = 0.3 * dfclust.imports ./ dfclust.price;  # slope
dfclust.am = dfclust.imports - dfclust.bm .* dfclust.price;  # intercept

describe(dfclust)
```

[3] ✓ 2.1s

As usual

12×7 DataFrame

Row	variable	mean	min	median	max	nmissing	eltype
	Symbol	Float64	Float64	Float64	Float64	Int64	DataType
1	price	38.5055	1.66311	37.2797	137.116	0	Float64
2	imports	7.52426	4.2479	7.57055	9.79814	0	Float64
3	q_commercial	12.9532	9.26236	12.3881	22.1766	0	Float64
4	q_industrial	4.1224	2.60049	3.83347	7.62003	0	Float64
5	q_residential	11.0753	4.3995	10.0075	20.4922	0	Float64

For this exercise, we use demand and a cost of $40/MWh of grid costs to estimate the annual cost of grid services. This is relevant to the issue of net-metering. If everyone goes off-grid, who pays for transmission?

```
grid_costs = sum(dfclust.weights .* dfclust.demand) * 40.0
```

✓ 0.0s Julia

```
9328.308818459258
```

The key of the “death spiral” is that there are some common costs (the grid) that are paid via the price. Households with solar panels can avoid contributing to some of these costs.

Adding net-metering

We will use a simplified version of the model with only solar panels.

All consumers will pay a flat tariff to make the issue of net metering more pervasive. There is also a certain amount of grid costs that need to be recovered.

Consumers will be able to net their demand out fully if `nem` equals 1. However, they will only receive the wholesale price if `nem` equals 0 for their non-coincident demand.

```
## Clear market based on first-order conditions
function clear_market_foc(data::DataFrame, tech::DataFrame;
    ng_price = 4.0, solve_invest=false,
    pretail = 50.0, kw = 3.0, hh = 12.0, nem = 1)

    # We declare a model
    model = Model(
        optimizer_with_attributes(
            HiGHS.Optimizer)
        );

    set_silent(model)

    # Set useful indexes
    I = nrow(tech);  # number of techs
    T = nrow(data);  # number of periods
    S = 3; # sectors
    M = 1e3;
```

We have a few more options: a flag to solve investment or keep it fixed, the retail price, the size of solar panels (kw) and the millions of households.
Nem is an indicator of the NEM policy.

Model has three sectors, focused on residential households

I assume that households consume proportionally to residential demand, and define if they are over or under the output from their panels. Selfc is self consumption, gout is the amount of solar put back in the grid.

```
# Defining self consumption
selfc = [minimum([data.solar_cap[t]*kw, data.q_residential[t]/hh]) for t=1:T];
gout  = [maximum([data.solar_cap[t]*kw - data.q_residential[t]/hh, 0.0]) for t=1:T]
```

```
@objective(model, Min, sum(price[t] * data.weights[t] for t=1:T));

# Market clearing
@constraint(model, [t=1:T], #residential demand inelastic to make calculation easier
    demand[t,1] == data.q_residential[t];
@constraint(model, [t=1:T,s=2:S],
    demand[t,s] == a[t,s] - b[t,s] * price[t]);
@constraint(model, [t=1:T],
    imports[t] == data.am[t] + data.bm[t] * price[t]);
@constraint(model, [t=1:T],
    sum(demand[t,s] for s=1:S) == sum(quantity[t,i] for i=1:I) + imports[t]);
```

This defines the profitability of different NEM policies.

```
if (solve_invest==true)
    # Definition of profit -- using quadratic costs here to get some more natural limits
    @constraint(model, [i=5:6], K[i]==0.0)
    if (nem==1) # always get retail price
        @constraint(model, profit[7] ==
            sum(data.weights[t]*pretail*data.solar_cap[t] for t=1:T) - tech.F[7] - 2 * tech.F2[7] * K[7]);
    elseif (nem==2) # cancel retail price if consuming, get wholesale price otherwise
        @constraint(model, profit[7] ==
            sum(data.weights[t]*(pretail*selfc[t] + shadow[t,7]*gout[t]) for t=1:T)/kw - tech.F[7] - 2 * tech.F2[7] * K[7]);
    elseif (nem==3) # cancel retail price if consuming, get zero otherwise
        @constraint(model, profit[7] ==
            sum(data.weights[t]*(pretail*selfc[t]) for t=1:T)/kw - tech.F[7] - 2 * tech.F2[7] * K[7]);
    else # wholesale investment
        @constraint(model, profit[7] ==
            sum(data.weights[t]*shadow[t,7] * data.solar_cap[t] for t=1:T) - tech.F[7] - 2 * tech.F2[7] * K[7]);
    end
    # Constraints on investment
    @constraint(model, [i=5:I], profit[i] >= -M*(1.0-u3[i])); # zero profits if investing
    @constraint(model, [i=5:I], K[i] <= M*u3[i]); # capacity only positive if firms can make zero profit
    @constraint(model, [i=5:I], profit[i] <= M*u4[i]) # profit zero unless at capacity
    @constraint(model, [i=5:I], K[i] >= kw*hh*u4[i]) # if u4 = 1, then capacity at max
else
    @constraint(model, [i=5:I], K[i]==0.0)
end
```

To keep things simple, this is only about solar panels.

I introduced an extra binary variable to “turn on” when all households already have panels.

```
res = clear_market_foc(dfclust,tech, kw=0.0)  # no solar panels
```
✓ 0.8s

```
Dict{String, Any} with 15 entries:
  "avg_price"     => 39.8063
  "price"         => [48.2163, 41.9634, 35.2882, 45.8763, 40.3516, 41.0625, 40.…
  "gas_gw"        => 0.0
  "avg_price_res" => 40.8914
  "status"        => "OPTIMAL"
  "u1"            => [1.0 1.0 … 1.0 1.0; 1.0 1.0 … 1.0 1.0; … ; 1.0 1.0 … 1.0 1…
  "quantity"      => [9.46003 11.5 … 0.0 0.0; 4.0713 11.5 … 0.0 0.0; … ; 2.6153…
  "solar_gw"      => 0.0
  "imports"       => [9.09234, 8.60823, 8.64448, 7.35063, 6.32325, 8.16486, 8.8…
  "demand"        => [13.9897 19.8277 4.13001; 13.4298 10.4243 2.75953; … ; 8.3…
  "shadow"        => [38.2163 17.2543 … 48.2163 48.2163; 31.9634 11.0015 … 41.9…
  "u2"            => [1.0 1.0 … 1.0 1.0; 1.0 1.0 … 1.0 1.0; … ; 1.0 1.0 … 1.0 1…
  "u3"            => 1-dimensional DenseAxisArray{Float64,1,...} with index set…
  "cost"          => 4622.31
  "wind_gw"       => 0.0
```

Households should pay much more to cover for the fixed costs of the grid.

Once we solve the model, we can see how much each household should pay: total costs divided by demand.

```
tariff = (sum(dfclust.weights[t] * res["price"][t] * dfclust.q_residential[t] for t=1:100)
            + grid_costs)/sum(dfclust.weights[t] * (dfclust.q_residential[t]) for t=1:100)
```
✓ 0.0s

```
141.36277900071903
```

Finding the equilibrium

We need to solve for the level of investment consistent with zero profits as well as the equilibrium tariff for consumers.

Investment is solved by the function, the tariff is solved outside the model in line with previous simulations.

Getting tariff as intermediate function

We create a function that solves the optimal tariff given how many consumers are contributing to recovery of grid costs, which depends on the net metering policy.

The function takes wholesale prices and solar investment as given.

The default recovers the previous result without netmetering.

Notice how net metering makes a very big difference.

Under net-metering, households not always contribute to pay: it erodes “the base” (the denominator).

Generate + Code + Markdown

```
function compute_tariff(data::DataFrame, prices, solar; solar_gw = 5.0, grid_cost = 10000.0, nem = 0, kw = 3.0, hh=12.0)

    T = nrow(data)

    data.selfc = [minimum([data.solar_cap[t]*kw, data.q_residential[t]/hh]) for t=1:T];
    data.gout  = [maximum([data.solar_cap[t]*kw - data.q_residential[t]/hh, 0.0]) for t=1:T]

    if (nem == 1) # consumers only pay for their non-solar demand, wholesale costs cancelled at wholesale price
        newguess = (sum(data.weights[t] * prices[t] * (data.q_residential[t]-data.selfc[t]*solar_gw/kw) for t=1:100) + grid_cost)/
                   sum(data.weights[t] * (data.q_residential[t]-data.solar_cap[t]*solar_gw) for t=1:T);
    elseif (nem == 2 || nem == 3) # consumers pay for all their non-coincidental demand
        newguess = (sum(data.weights[t] * prices[t] * (data.q_residential[t]-data.selfc[t]*solar_gw) for t=1:100) + grid_cost)/
                   sum(data.weights[t] * (data.q_residential[t]-data.selfc[t]*solar_gw/kw) for t=1:T);
    else # all consumers pay for all costs (non-residential investment)
        newguess = (sum(data.weights[t] * prices[t] * (data.q_residential[t]-data.selfc[t]*solar_gw/kw) for t=1:100) + grid_cost)/
                   sum(data.weights[t] * (data.q_residential[t]) for t=1:100);
    end
end
```

[8] ✓ 0.0s

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

Without net-metering, everyone pays for their grid costs in proportion to their gross consumption.

```
net0 = compute_tariff(dfclust, res["price"], res["quantity"][:,7], kw=3.0, grid_cost=grid_costs, nem=0)
```

9] ✓ 0.1s

```
139.10399074451084
```

Under net-metering 1.0, solar customers only pay net of their solar production. Thus, the tariff needs to be higher.

```
net1 = compute_tariff(dfclust, res["price"], res["quantity"][:,7], kw=3.0, grid_cost=grid_costs, nem=1)
```

2] ✓ 0.1s

```
152.40984476273442
```

Under net-metering 2.0, we get a middle ground. Solar customers only pay for their consumption net of self-consumption. The residential tariff is in-between the two cases.

```
net2 = compute_tariff(dfclust, res["price"], res["quantity"][:,7], kw=3.0, grid_cost=grid_costs, nem=2)
```

7] ✓ 0.1s

```
141.82986443563786
```

Notice, this is holding investment fixed! Incentives to invest will be affected by these tariffs.

Now we compute the market equilibrium by solving investment and guessing tariff

How does the net-metering scheme affect total investment? By allowing customers to net out their grid costs, net-metering 1.0 can make solar panels quite attractive.

To solve the equilibrium, we use the "solve_invest" function in the formula, and iterate to look for the equilibirum tariff, using the compute tariff function to know how much the tariff should be.

```
function clear_market_equilibrium(data::DataFrame, tech::DataFrame;
    grid_cost = 10000.0, pretail=150.0, kw=3.0, hh=12.0, ng_price=4.0, solve_invest=true, nem = 1)

    current_diff = 1.0;
    guess = pretail;
    while (current_diff > 1e-2)
        res = clear_market_foc(data, tech, solve_invest=solve_invest,
                pretail=guess, kw=kw, hh=hh, ng_price=ng_price, nem=nem);
        if (res["status"]=="OPTIMAL")
            newguess = compute_tariff(dfclust, res["price"], res["quantity"][:,7],
                                solar_gw=res["solar_gw"], nem=nem, grid_cost=grid_cost)
        else
            @printf "No solution, status %s, " res["status"]
            @printf "tariff guess %f, " guess
            newguess = guess;
        end
        current_diff = (guess-newguess).^2;
        guess = newguess;
    end

    # we solve at the equilibrium guess
    res = clear_market_foc(dfclust, tech, solve_invest=solve_invest,
        pretail=guess, kw=kw, hh=hh, ng_price=ng_price, nem=nem);
    res["pretail"] = guess;

    return res

end
```

37] ✓ 0.0s

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

Here we find the equilibrium investment.
We update retail prices and resolve the market equilibrium.

Under net-metering 1.0, consumers can cancel their grid payments when producing with their solar power, regardless of whether they consume at the same time or not. Thus, we get more investment. It has great impacts on the retail price of consumers "left behind".

In this code, we only have one type of consumer, but in practice, some HHs put solar panels, and some don't, leading to very different retail prices.

```julia
res_eq1 = clear_market_equilibrium(dfclust, tech, solve_invest=true, nem = 1)
res_eq1["solar_gw"]/3.0
```

✓ 0.4s Julia

```
12.0
```

The final tariffs are much different!
With NEM 1, here everyone invested, so no one left behind, but in practice households cannot access these investments.

```julia
res_eq1["pretail"]
```

✓ 0.0s Julia

```
354.7564496727337
```

Under net-metering 2.0, consumers can only cancel their grid payments when producing with their solar power and consuming at the same time. Net-metering 1.0 acts as an implicit subsidy to solar panels.

```julia
res_eq2 = clear_market_equilibrium(dfclust, tech, solve_invest=true, nem = 2)
res_eq2["solar_gw"]/3.0
```

✓ 0.1s Julia

```
5.524658807093846
```

```julia
res_eq2["pretail"]
```

✓ 0.0s Julia

```
153.17957643412956
```

Residential solar can be more profitable than non-residential solar. For the equivalent non-residential solar outcome, we set nem to 0. In this case, we obtain zero investment without the additional incentive to avoid grid costs.

In practice, large scale solar is substantially cheaper, so both can be profitable.

```
res_eq0 = clear_market_equilibrium(dfclust, tech, solve_invest=true, nem = 0)
res_eq0["solar_gw"]/3.0
```

[6] ✓ 0.1s Julia

```
0.0
```

```
res_eq0["pretail"]
```

[7] ✓ 0.0s Julia

```
148.59728979588064
```

Tariff design is a way to encourage residential rooftop solar. The equilibrium can be nuanced and still a source of public debate.