Day 8: Demand - Adding insensitive consumers

We will include consumers that fail to respond to real-time prices, to examine how that impacts welfare and short/long-run outcomes. This will be in line with Borenstein and Holland (2005).

The data and code are based on the paper "The Efficiency and Sectoral Distributional Implications of Large-Scale Renewable Policies," by Mar Reguant.

The code builds on Day 5 (tariff setting).

We first load relevant libraries, same as last session.

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

from pyomo.environ import *
```

5] ✓ 0.0s Python

Remember to set your path correctly:

```
dirpath = "/Users/marreguant/Dropbox/TEACHING/BSE/Electricity2026/day8/practicum/"
```

7] ✓ 0.0s Python

Building the model

We load the same data as last week, and also clean it up to simplify it further and create the demand and import curves. We annualize fixed costs for new plants.

```
TECHS = ["hydronuc","gas1","gas2","gas3","newgas","wind","solar"]
INV_TECHS = ["newgas","wind","solar"]
RES_TECHS = ["wind","solar"]
```

0.0s Python

```
def load_and_prepare(dirpath: str):

    dfclust = pd.read_csv(f"{dirpath}/data_jaere_clustered.csv")
    tech = pd.read_csv(f"{dirpath}/data_technology.csv")

    # Re-scaling weights: multiply by 8.76 and normalize (so that we can interpret it as annual revenue in million USD)
    dfclust["weights"] = 8.76 * dfclust["weights"] / dfclust["weights"].sum()


    # Calibrate imports: imports = am + bm * price
    elas = np.array([0.1, 0.2, 0.5, 0.3])
    dfclust["bm"] = elas[3] * dfclust["imports"] / dfclust["price"]
    dfclust["am"] = dfclust["imports"] - dfclust["bm"] * dfclust["price"]

    # Set index names for tech
    tech.index = TECHS
```

We will calibrate demand in the main function to allow for different elasticities

Adding insensitive demand to the problem

To include demand, we will separate the price that consumers pay from the price that the wholesale market sets. We define a variable "tariff" that is what consumers pay any hour of the day.

We also allow alpha to determine the share of "insensitive" consumers. We can see how the investment changes as we modify "alpha".

By popular demand, this code also include three different sectors with different elasticities and degrees of sensitivity.

```
def clear_market_invest(data, tech, ng_price=3.5, tax=0.0, tariff=[40.0, 40.0, 40.0],
                elas = [.1, .2, .5, .3], alpha = [1.0, 1.0, 1.0],
                Kmax=50.0, solver_name="ipopt"):
    """
    Welfare max with endogenous investment (NLP)
    """
    T = len(data)
    S = 3 # demand segments

    # plain python lookups (no Params)
    c = {k: float(tech.at[k, "c"]) for k in TECHS}
    if "heatrate" in tech.columns:
        for k in ["gas1","gas2","gas3","newgas"]:
            if k in tech.index:
                c[k] = float(tech.at[k, "heatrate"]) * float(ng_price)

    e = {k: float(tech.at[k, "e"]) for k in TECHS} if "e" in tech.columns else {k: 0.0 for k in TECHS}
    F = {k: float(tech.at[k, "F"]) for k in TECHS} if "F" in tech.columns else {k: 0.0 for k in TECHS}
    capUB = {k: float(tech.at[k, "capUB"]) for k in ["gas1","gas2","gas3"] if "capUB" in tech.columns and k in tech.index}
```

We have new options: tariffs, elasticities, and sensitive share (alpha)

We have three consumer types

```python
# assume T = number of rows (time), S = number of demand segments
a = np.zeros((T, S))
b = np.zeros((T, S))

p = np.asarray(data["price"])  # or data.price if it's an object with attributes

# Helper to keep things readable
def calibrate_linear_demand(q, elasticity, price):
    """
    Calibrate q = a - b*price with constant elasticity at the observed point:
    elasticity = (dq/dp) * (p/q)  =>  dq/dp = elasticity * (q/p)
    We store slope 'b' (positive) so demand is q = a - b*p.
    """
    q = np.asarray(q)
    slope = elasticity * q / price          # pointwise dq/dp (likely negative)
    slope = np.mean(slope) * np.ones_like(q) # make slope constant over t
    intercept = q + slope * price            # because q = a - b*p and slope is dq/dp
    return intercept, slope

# Residential
a[:, 0], b[:, 0] = calibrate_linear_demand(
    q=data["q_residential"], elasticity=elas[0], price=p
)

# Commercial
a[:, 1], b[:, 1] = calibrate_linear_demand(
    q=data["q_commercial"], elasticity=elas[1], price=p
)

# Industrial
a[:, 2], b[:, 2] = calibrate_linear_demand(
    q=data["q_industrial"], elasticity=elas[2], price=p
)
```

This part calibrates the demand for each segment of consumers

```
m = ConcreteModel()
m.T = RangeSet(0, T-1)
m.S = RangeSet(0, S-1)
m.I = Set(initialize=TECHS, ordered=False)
m.J = Set(initialize=INV_TECHS, ordered=False)

m.price   = Var(m.T, domain=Reals)
m.demand  = Var(m.T, m.S, domain=Reals)
m.imports = Var(m.T, domain=Reals)

m.q     = Var(m.T, m.I, domain=NonNegativeReals)
m.costs = Var(m.T, domain=Reals)
m.gs    = Var(m.T, domain=Reals)

m.K      = Var(m.J, bounds=(0.0, float(Kmax)))  # capacity investments

# Objective: gross surplus - variable costs - fixed costs
m.obj = Objective(
    expr=sum(data.weights[t] * (m.gs[t] - m.costs[t]) for t in m.T)
        - sum(F.get(k) * m.K[k] for k in m.J),
    sense=maximize
)

# Market clearing
m.demand_curve  = Constraint(m.T, m.S, rule=lambda m,t,s:
    m.demand[t,s]  == a[t,s] - b[t,s] * (alpha[s] * m.price[t] + (1.0 - alpha[s]) * tariff[s]))
m.imports_curve = Constraint(m.T, expr={t: m.imports[t] == data.am[t] + data.bm[t] * m.price[t] for t in m.T})
m.market_clear  = Constraint(m.T, expr={t: sum(m.demand[t, s] for s in m.S)  == sum(m.q[t,k] for k in m.I) + m.imports[t] for
```

There is a new index (consumers)

Demand is the combination of two types

```python
# Surplus + costs
m.surplus_def = Constraint(
    # Note: now surprlus only matters for sensitive households (rest fixed)
    m.T, rule = lambda m,t:  m.gs[t] ==                    Gross surplus only includes sensitive part
    sum(alpha[s]*((b[t,s] * m.price[t]) * (a[t,s] -  b[t,s] * m.price[t]) / b[t,s]
                  + ((a[t,s] - b[t,s] * m.price[t])**2) / (2.0 * b[t,s])) for s in m.S)
)
m.cost_def = Constraint(
    m.T, rule = lambda m,t: m.costs[t] == sum((c[k] + float(tax) * e[k]) * m.q[t,k] for k in m.I)
                    + (m.imports[t] - data.am[t])**2 / (2.0 * data.bm[t])
)

# Capacity constraints
m.cap_hydronuc = Constraint(m.T, expr={t: m.q[t,"hydronuc"] <= data.hydronuc[t] for t in m.T})
if capUB:
    m.cap_gas123 = Constraint(m.T, Set(initialize=list(capUB)),
                              expr={(t,k): m.q[t,k] <= capUB[k] for t in m.T for k in capUB})

m.cap_newgas = Constraint(m.T, expr={t: m.q[t,"newgas"] <= m.K["newgas"] for t in m.T})
m.cap_wind   = Constraint(m.T, expr={t: m.q[t,"wind"]   <= m.K["wind"]  * data.wind_cap[t]  for t in m.T})
m.cap_solar  = Constraint(m.T, expr={t: m.q[t,"solar"]  <= m.K["solar"] * data.solar_cap[t] for t in m.T})

res = SolverFactory(solver_name).solve(m, tee=False)
term = str(res.solver.termination_condition)
```

```python
if term.lower() in ("optimal", "locallyoptimal", "locally_optimal", "locally optimal"):
    price   = np.array([value(m.price[t])   for t in m.T])
    demand  = np.array([value(m.demand[t,s])  for t in m.T for s in m.S]).reshape((T, S))
    imports = np.array([value(m.imports[t]) for t in m.T])
    q = {k: np.array([value(m.q[t,k]) for t in m.T]) for k in TECHS}
    K = {k: float(value(m.K[k]))           for k in INV_TECHS}

    w_arr = data["weights"].to_numpy()
    avg_price = float(np.sum(price * w_arr) / np.sum(w_arr))

    objective = float(np.sum([data.weights[t] * (value(m.gs[t]) - value(m.costs[t])) for t in m.T])
                      - F.get("newgas", 0.0) * K["newgas"]
                      - F.get("wind",   0.0) * K["wind"]
                      - F.get("solar",  0.0) * K["solar"])

    w_arr = data["weights"].to_numpy()
    emissions = sum(w_arr[t] * sum(e[i] * q[i][t] for i in m.I) for t in m.T)
    avg_price = float(np.sum(price * w_arr)/np.sum(w_arr))
    demand_insen = [a[t, :] - b[t, :] * tariff for t in range(T)]
    num = (w_arr[:, None] * price[:, None] * demand_insen).sum(axis=0)
    den = (w_arr[:, None] * demand_insen).sum(axis=0)
    avg_customer_cost = num / den

    # cost accounting
    gen_cost_t = np.array([value(m.costs[t])   for t in m.T])
    imp_cost_t = (imports - data["am"].to_numpy()) ** 2 / (2.0 * data["bm"].to_numpy())
    total_cost = float(np.sum(w_arr * (gen_cost_t + imp_cost_t))) + F.get("newgas", 0.0) * K["newgas"] +  F.get("wind",   0.0) * K

    return {"status": term, "avg_price": avg_price, "price": price,
            "quantity": q, "imports": imports, "avg_customer_cost": avg_customer_cost,  "total_cost": total_cost,
            "demand": demand, "cost": total_cost, "objective": objective, "K": K, "emissions": emissions}
```

I calculate the weighted price of insensitive consumers to understand how much they should pay as a tariff to cover their wholesale market costs.

```
res_sensitive = clear_market_invest(dfclust, tech, ng_price=3.5, alpha=[1.0, 1.0, 1.0])
```

0.2s Python

```
res_sensitive["avg_customer_cost"]
```

0.0s Python

```
array([33.29755455, 33.32863139, 34.56719396])
```

Average price if consumers are sensitive

```
res_insensitive = clear_market_invest(dfclust, tech, ng_price=3.5, alpha=[0.1, 0.1, 0.1], tariff=res_sensitive["avg_customer_cost"])
```

0.2s Python

```
res_insensitive["avg_customer_cost"]
```

0.0s Python

```
array([35.02143549, 34.9013057 , 37.24577261])
```

If we charge consumers the average price under ideal conditions but consumers are not sensitive, then the true cost is higher! We need to change the tariffs.

We can see that having inattentive consumers makes costs higher. We can also see there is further need for investment.

```
res_sensitive["K"]
```

✓ 0.0s

```
{'newgas': 5.9552191703331445,
 'wind': -8.540889949828152e-09,
 'solar': -9.933335474962674e-09}
```

```
res_insensitive["K"]
```

✓ 0.0s

```
{'newgas': 7.386339885974402,
 'wind': -7.800055260514229e-09,
 'solar': -9.91607384301638e-09}
```

Unresponsive demand leads to inefficient investment

We create a function that will do the loop and return the average price that we need to charge consumers of each kind.

```
def clear_market_equilibrium(data, tech, ng_price=3.5, tax=0.0,
                             tariff = [40.0,40.0,40.0], elas = [.1, .2, .5, .3], alpha = [0.2, 0.3, 0.8],
                             tol=1e-3, max_iter=10):
    current_diff = 1.0
    guess = tariff
    it = 0

    while current_diff > tol and it < max_iter:
        res = clear_market_invest(
            data, tech,
            ng_price=ng_price, tax=tax, tariff=guess, elas=elas, alpha=alpha
        )

        if res["status"] == "optimal":
            newguess = res["avg_customer_cost"]
        else:
            print(f"Model is {res['status']} at {guess}")
            return res

        current_diff = sum((guess - newguess) ** 2)
        guess = newguess
        it += 1

    # solve at (approx) equilibrium to return results
    res = clear_market_invest(
        data, tech,
        ng_price=ng_price, tax=tax, tariff=guess, elas=elas, alpha=alpha
    )
    return res
```

] ✓ 0.0s

This function ensures that we charge consumers their weighted average price (avg_customer_cost).

```
res_eq = clear_market_equilibrium(dfclust,tech, alpha=[0.1, 0.1, 0.1])
```

0] ✓ 0.8s

```
print(res_eq["avg_customer_cost"])
```

1] ✓ 0.0s

```
[35.0302578  34.92655045 37.51806004]
```

Equilibrium final prices need to be higher, indeed, very similar to first guess as consumers are not very elastic, although it already impacts investment effects a bit.

```
res_eq["K"]
```

✓ 0.0s

```
{'newgas': 7.02966292485785,
 'wind': -7.850735807045468e-09,
 'solar': -9.916076489522443e-09}
```

Does it help to have attentive consumers?

We visualize the impact of the policy of changing residential customers sensitivity.

We can plot the costs (or prices) of electricity for several values of alpha. This can take a while to compute.

```
# We create a grid to produce results under different responsiveness
share_alpha = np.arange(0.0, 1.0 + 1e-9, 0.2)   # 0.0,0.2,...,1.0
tax_grid    = np.arange(0.0, 30.0 + 1e-9, 15.0) # 0,15,30
elas_grid   = np.arange(0.1, 0.5 + 1e-9, 0.2)   # 0.1,0.3,0.5

rows = []
for a in share_alpha:
    for t in tax_grid:
        for e in elas_grid:
            res = clear_market_equilibrium(
                dfclust, tech,
                ng_price=3.5,
                alpha=[a, a, a],
                elas=[e, 0.2, 0.5, 0.3],
                tax=t
            )
            rows.append({
                "alpha": float(a),
                "elas": float(e),
                "tax": float(t),
                "tariff_res": float(res["avg_customer_cost"][0]),
                "tariff_com": float(res["avg_customer_cost"][1]),
                "tariff_ind": float(res["avg_customer_cost"][2]),
                "gas_gw": float(res["K"]["newgas"]),
                "wind_gw": float(res["K"]["wind"]),
                "solar_gw": float(res["K"]["solar"]),
            })

dataPolicy = pd.DataFrame(rows)
```

This code can help you see how to compute alternative “counterfactuals” using your model and storing them into a data frame for further exploration later

```
dataPolicy.head(20)
```

✓ 0.0s

	alpha	elas	tax	tariff_res	tariff_com	tariff_ind	gas_gw	wind_gw	solar_gw
0	0.0	0.1	0.0	36.106842	36.087823	40.075110	7.720631e+00	-6.986181e-09	-9.910062e-09
1	0.0	0.3	0.0	38.866532	36.543490	41.194971	9.116417e+00	-7.765995e-09	-9.907218e-09
2	0.0	0.5	0.0	43.350850	36.963346	42.160289	9.885999e+00	-7.488196e-09	-9.908721e-09
3	0.0	0.1	15.0	40.159069	40.583194	45.052464	3.425700e+00	1.242698e+01	-9.878969e-09
4	0.0	0.3	15.0	43.446897	41.209956	46.908527	4.515498e+00	1.092501e+01	-9.872289e-09
5	0.0	0.5	15.0	48.786090	41.513841	47.926991	6.042409e+00	7.676316e+00	-9.877462e-09
6	0.0	0.1	30.0	42.325245	43.208350	48.459796	9.497440e-01	2.054921e+01	-9.536702e-09
7	0.0	0.3	30.0	45.845545	43.669161	50.459064	2.291067e+00	1.764236e+01	-9.371245e-09
8	0.0	0.5	30.0	52.810693	43.865279	51.959774	3.527611e+00	1.285030e+01	-9.472643e-09
9	0.2	0.1	0.0	34.507647	34.394339	36.467816	6.724167e+00	-8.097521e-09	-9.921824e-09
10	0.2	0.3	0.0	35.796707	34.476497	36.813742	6.943329e+00	-8.353595e-09	-9.921439e-09
11	0.2	0.5	0.0	37.461132	34.507437	37.102876	6.936856e+00	-8.492827e-09	-9.921742e-09
12	0.2	0.1	15.0	39.154622	39.216865	41.476429	3.103044e+00	1.116336e+01	-9.901654e-09
13	0.2	0.3	15.0	40.575810	39.252594	41.898025	3.216258e+00	1.050672e+01	-9.896107e-09
14	0.2	0.5	15.0	42.587051	39.233236	42.262457	3.005789e+00	9.801395e+00	-9.896751e-09
15	0.2	0.1	30.0	41.570508	41.973610	44.776954	1.245192e-01	2.199172e+01	-9.868567e-09
16	0.2	0.3	30.0	43.299863	42.012387	45.307743	1.021501e-02	2.059439e+01	-9.867151e-09
17	0.2	0.5	30.0	45.619835	41.787971	45.507346	1.124520e-07	1.826859e+01	-9.868392e-09
18	0.4	0.1	0.0	33.921492	33.817481	35.380132	6.474060e+00	-8.392373e-09	-9.928126e-09
19	0.4	0.3	0.0	34.781105	33.856119	35.558758	6.712862e+00	-8.465927e-09	-9.928254e-09

The dataframe stores the relevant policy outcomes that I decided to focus on along the relevant margins I decided to change (here alpha, elasticity, and the carbon tax)

We can plot how wind investment is affected by having "flexible demand".

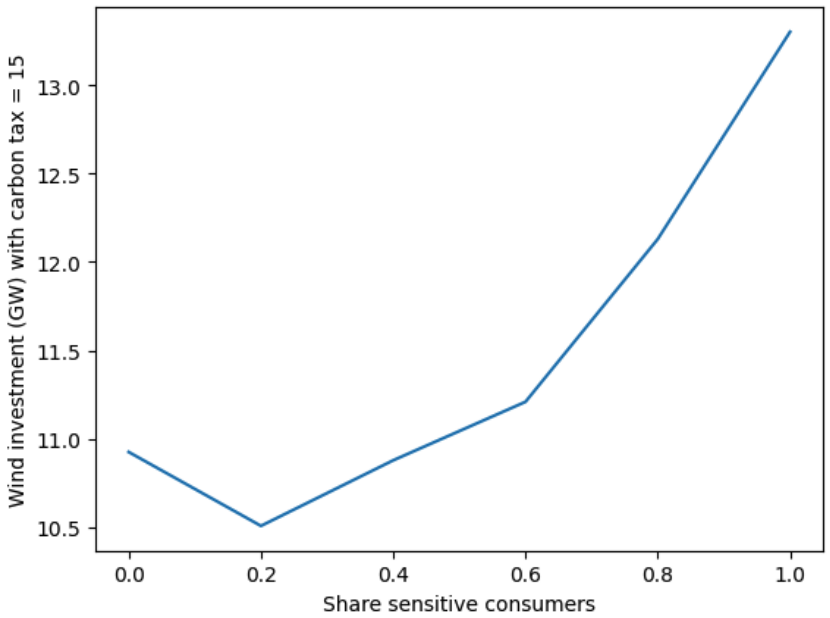
```
# We select a certain "slice" of the data to plot
# Note: we round to avoid floating point issues

carbontax = 15
elasticity = 0.3

tax1 = dataPolicy[(dataPolicy["tax"].round(0) == carbontax) & (dataPolicy["elas"].round(1) == elasticity)].copy()
tax1 = tax1.sort_values("alpha")

plt.figure()
plt.plot(tax1["alpha"], tax1["wind_gw"])
plt.xlabel("Share sensitive consumers")
plt.ylabel(f"Wind investment (GW) with carbon tax = {carbontax}")
plt.show()
```

✓ 0.0s

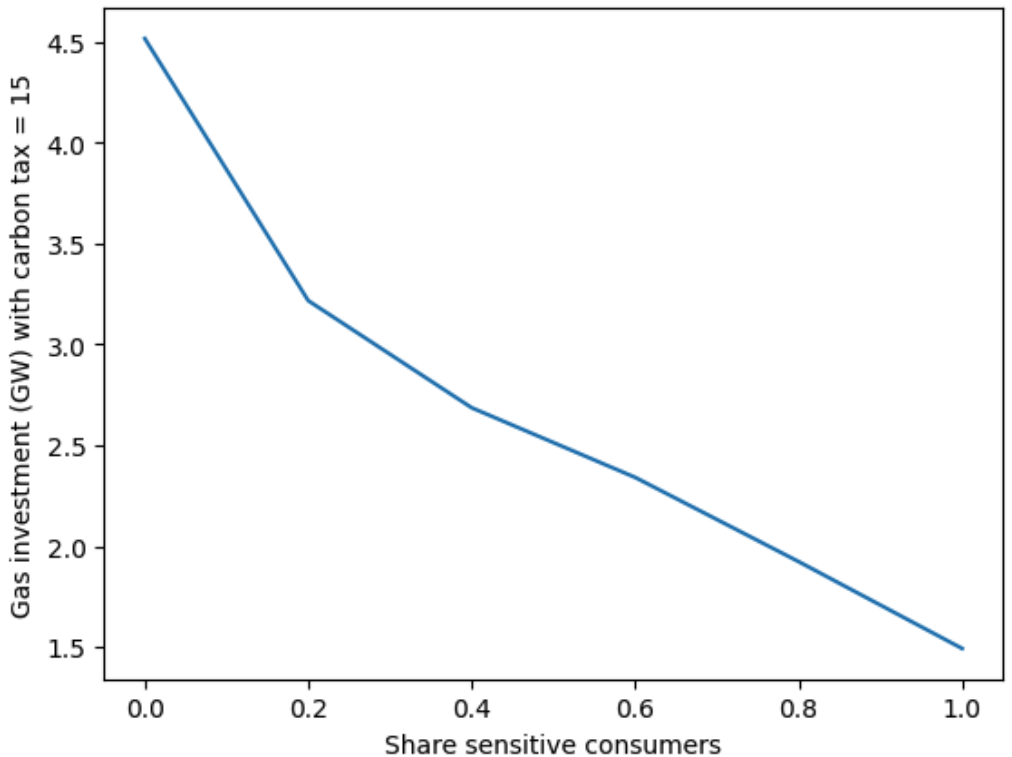


Investment in wind is increasing in the share of sensitive consumers.

When there is a carbon tax, more sensitive consumers help support more renewable power (holding the tax fixed!)

```
plt.figure()
plt.plot(tax1["alpha"], tax1["gas_gw"])
plt.xlabel("Share sensitive consumers")
plt.ylabel(f"Gas investment (GW) with carbon tax = {carbontax}")
plt.show()
```

✓ 0.0s

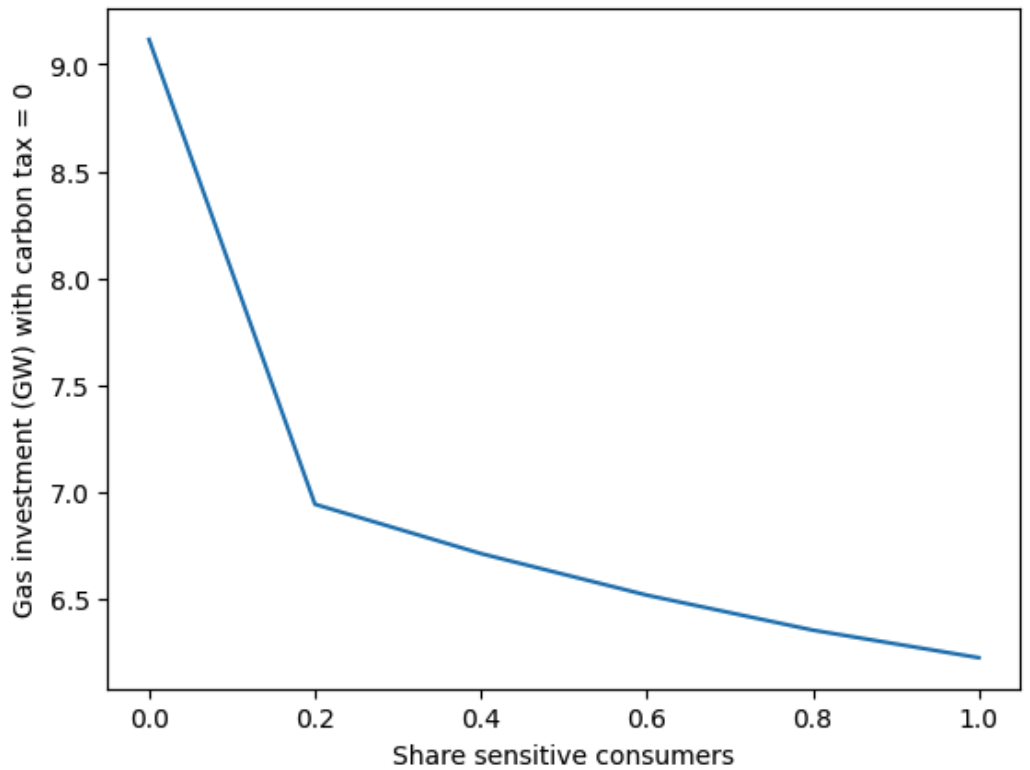


When there is a carbon tax, more sensitive consumers help reduce gas investment.

```
tax2 = dataPolicy[(dataPolicy["tax"].round(0) == 0) & (dataPolicy["elas"].round(1) == elasticity)].copy()
tax2 = tax2.sort_values("alpha")

plt.figure()
plt.plot(tax2["alpha"], tax2["gas_gw"])
plt.xlabel("Share sensitive consumers")
plt.ylabel(f"Gas investment (GW) with carbon tax = 0")
plt.show()
```

✓ 0.0s



When there is no carbon tax, more sensitive consumers tend to reduce the need for thermal investment, although this is not guaranteed, it is what models tend to find (Thm 4 in Borenstein and Holland, 2005).