BEE 4750 Homework 4: Generating Capacity Expansion

> Due Date
>
> Friday, 10/27/23, 9:00pm

Overview

Instructions

- In Problem 1, you will formulate, solve, and analyze a standard generating capacity expansion problem.
- In Problem 2, you will add a CO_2 constraint to the capacity expansion problem and identify changes in the resulting solution.

Load Environment

The following code loads the environment and makes sure all needed packages are installed. This should be at the start of most Julia scripts.

```
import Pkg
Pkg.activate(@__DIR__)
Pkg.instantiate()
```

```
using JuMP
using HiGHS
using DataFrames
using Plots
using Measures
using CSV
using MarkdownTables
```

Problems (Total: 100 Points)

For this problem, we will use hourly load (demand) data from 2013 in New York's Zone C (which includes Ithaca). The load data is loaded and plotted below.

```
# load the data, pull Zone C, and reformat the DataFrame
NY_demand = DataFrame(CSV.File("data/2013_hourly_load_NY.csv"))
rename!(NY_demand, :"Time Stamp" => :Date)
demand = NY_demand[:, [:Date, :C]]
rename!(demand, :C => :Demand)
demand[:, :Hour] = 1:nrow(demand)

# plot demand
plot(demand.Hour, demand.Demand, xlabel="Hour of Year", ylabel="Demand (MWh)", label=:false)
```

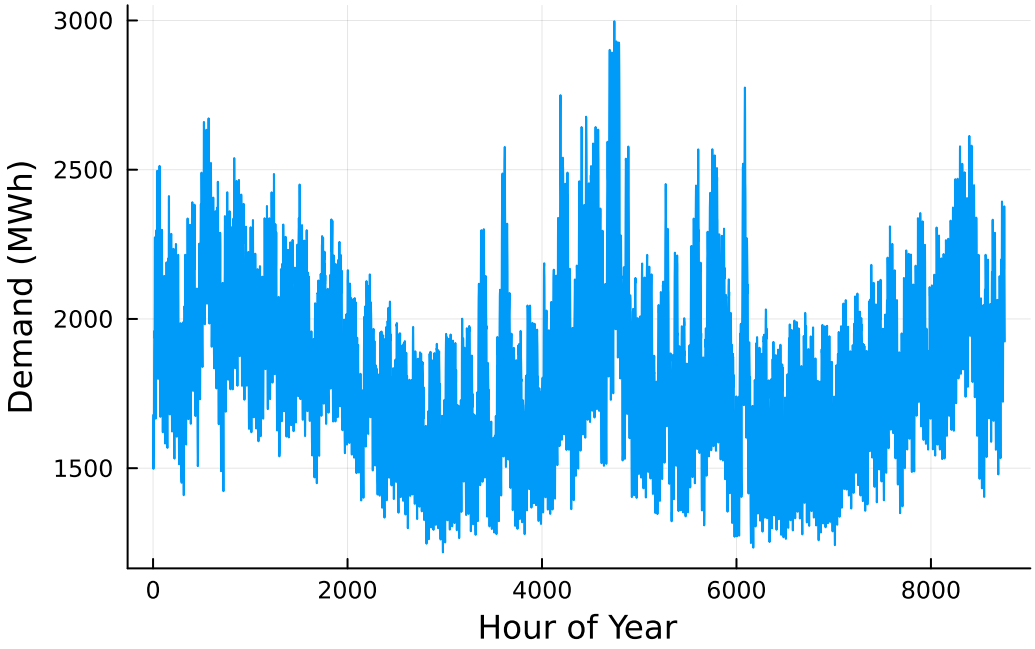


Figure 1: Hourly load data for New York's Zone C in 2013.

Next, we load the generator data. This data includes fixed costs ($/MW installed), variable costs ($/MWh generated), and CO_2 emissions intensity (tCO_2/MWh generated).

```
gens = DataFrame(CSV.File("data/generators.csv"))
```

Table 1: Generator type data, including fixed costs ($/MW installed$), $variable costs$(/MWh generated), and CO_2 emissions intensity (tCO_2/MWh generated).

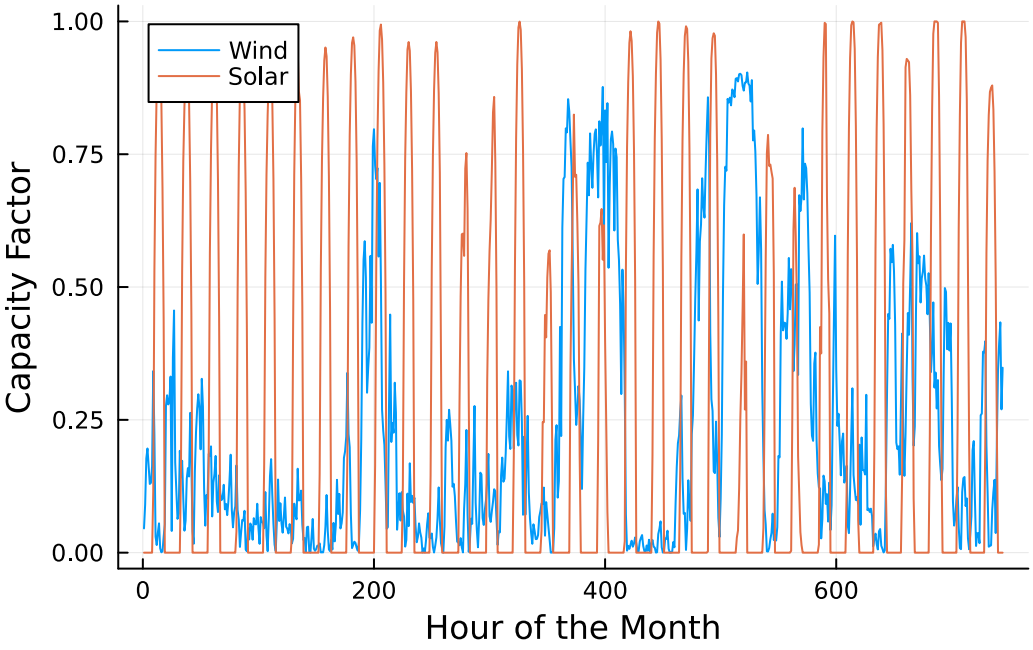
	Plant	FixedCost	VarCost	Emissions
	String15	Int64	Int64	Float64
1	Geothermal	450000	0	0.0
2	Coal	220000	24	1.0
3	NG CCGT	82000	30	0.43
4	NG CT	65000	40	0.55
5	Wind	91000	0	0.0
6	Solar	70000	0	0.0

Finally, we load the hourly solar and wind capacity factors, which are plotted in Figure 2. These tell us the fraction of installed capacity which is expected to be available in a given hour for generation (typically based on the average meteorology).

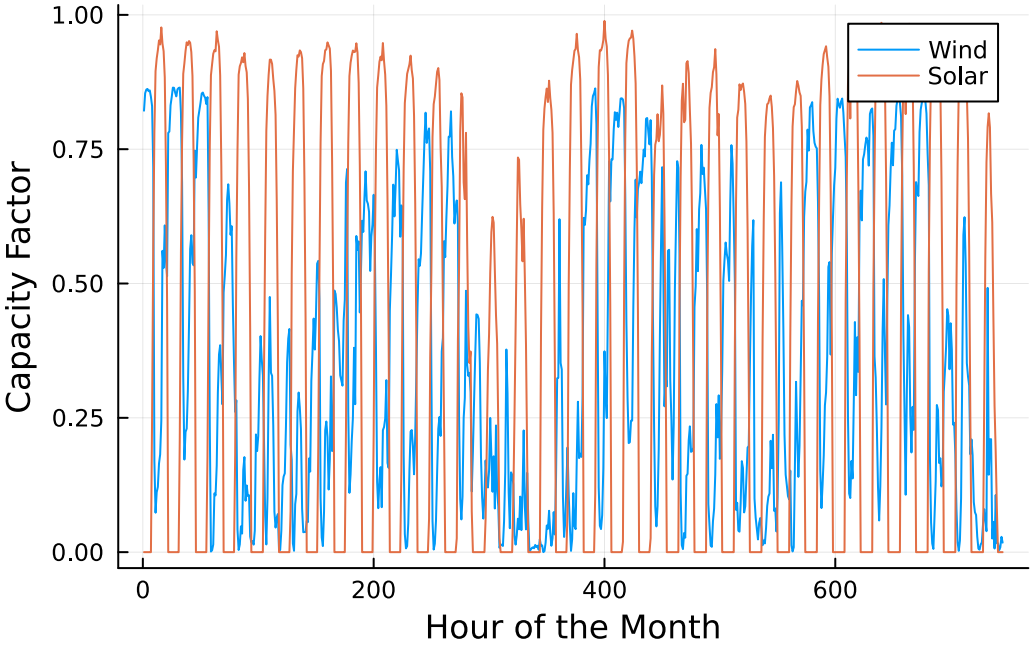
```
# load capacify factors into a DataFrame
cap_factor = DataFrame(CSV.File("data/wind_solar_capacity_factors.csv"))

# plot January capacity factors
p1 = plot(cap_factor.Wind[1:(24*31)], label="Wind")
plot!(cap_factor.Solar[1:(24*31)], label="Solar")
xaxis!("Hour of the Month")
yaxis!("Capacity Factor")

p2 = plot(cap_factor.Wind[4344:4344+(24*31)], label="Wind")
plot!(cap_factor.Solar[4344:4344+(24*31)], label="Solar")
xaxis!("Hour of the Month")
yaxis!("Capacity Factor")

display(p1)
display(p2)
```

You have been asked to develop a generating capacity expansion plan for the utility in Riley, NY, which currently has no existing electrical generation infrastructure. The utility can build any of the following plant types: geothermal, coal, natural gas combined cycle gas turbine (CCGT), natural gas combustion turbine (CT), solar, and wind.

While coal, CCGT, and CT plants can generate at their full installed capacity, geothermal plants operate at maximum 85% capacity, and solar and wind available capacities vary by the hour depend on the expected meteorology. The utility will also penalize any non-served demand at a rate of $1000/MWh.



(a) January



(b) July

Figure 2: Hourly solar and wind capacity factors.

> **Significant Digits**
>
> Use `round(x; digits=n)` to report values to the appropriate precision!

> **Getting Variable Output Values**
>
> `value.(x)` will report the values of a `JuMP` variable `x`, but it will return a special container which holds other information about `x` that is useful for `JuMP`. This means that you can't use this output directly for further calculations. To just extract the values, use `value.(x).data`.

Problem 1 (22 points)

Your first task is to find a capacity expansion plan which minimizes total costs of investment and operation.

Problem 1.1 (2 points)

Identify and define the decision variables for the problem. Make sure to include units.

Solution:

We need the following notation:

- x_g: built capacity for generator g in MW;
- $y_{g,t}$: generated power from generator g in time period t in MWh;
- NSE_t: unserved energy in time t.

Problem 1.2 (3 points)

Formulate the objective function. Make sure to include any needed derivations or justifications for your equation(s) and define any additional required notation beyond that introduced in Problem 1.1.

Solution:

The objective is to minimize total costs (C) of expansion and generation, which we can split into fixed costs (FixedCost, \$/MW installed), variable costs of generation (\$/MWh generated), and the non-served energy penalty (\$/MWh unserved).

We can write this as

$$\min_{x_g, y_{g,t}} \sum_{g \in \mathcal{G}} \text{FixedCost}_g \times x_g + \sum_{t \in \mathcal{T}} \text{VarCost}_g \times y_{g,t} + \sum_{t \in \mathcal{T}} \text{NSECost}_t \times NSE_t,$$

where $\mathcal{G}$ is the set of generators and $\mathcal{T}$ is the set of time periods.

Problem 1.3 (4 points)

Derive all relevant constraints. Make sure to include any needed justifications or derivations.

Solution:

The first constraint is non-negativity; all decision variables must be greater than or equal to zero, so $x_g \geq 0$, $y_{g,t} \geq 0$, and $NSE_t \geq 0$.

Second, the amount of generated power is limited by the capacity of that plant type. This is slightly more complicated because we need to account for the variable capacity factors of the wind and solar plants. We can summarize this with

$$y_{g,t} \leq x_g \times cf_{g,t}$$

for all $t \in \mathcal{T}$, where $cf_{g,t}$ is the capacity factor for plant type g in time t. We can either just assume that $cf_{g,t} = 1$ for non-variable plants g (other than geothermal, where $cf = 0.85$) or we can split this into multiple constraints.

Third, the sum of generated power and un-served energy must be equal to the demand in that time period:

$$\sum_{g \in G} y_{g,t} + NSE_t = d_t$$

for all $t \in \mathcal{T}$, where d_t is the demand in MWh.

Problem 1.4 (3 points)

Implement your optimization problem in `JuMP`.

Solution:

As mentioned above, we can either split the capacity constraint into multiple statements, or we can set up a single capacity factor matrix which combines the variable wind and solar capacity factors and constant values of 1 for the other plant types. We will do the second in this solution, but the first would get you the same answers (you would just have one-two more constraints defined, depending on the specifics).

```
# define sets
G = 1:nrow(gens)
T = 1:nrow(demand)

# set up auxiliary parameters
NSECost = 1000 # cost of unserved energy
```

```julia
# capacity factor matrix
cf_constant = ones(T[end], G[end-2]) # last two plants are wind and solar, so drop those
cf_constant[:, 1] .= 0.85
cf = hcat(cf_constant, cap_factor[!, :Wind], cap_factor[!, :Solar]) # append wind and solar c


# set up model object
gencap = Model(HiGHS.Optimizer) # use the HiGHS LP solver

# define variables
@variable(gencap, x[g in G] >= 0) # installed capacity
@variable(gencap, y[g in G, t in T] >= 0) # generated power
@variable(gencap, nse[t in T] >= 0) # unserved energy

# define objective: minimize sum of fixed costs, variable costs of generation,
# and non-served energy penalty
@objective(gencap, Min, sum(gens[:, :FixedCost] .* x) +
    sum(gens[:, :VarCost] .* [sum(y[g, :]) for g in G]) +
    NSECost * sum(nse))

# define constraints
@constraint(gencap, capacity[g in G, t in T], y[g, t] <= x[g] * cf[t, g]) # capacity constrai
@constraint(gencap, demand_met[t in T], sum(y[:, t]) + nse[t] == demand.Demand[t]) # demand c
```

```
1-dimensional DenseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInt
    Dimension 1, 1:8760
And data, a 8760-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOpt
 demand_met[1] : y[1,1] + y[2,1] + y[3,1] + y[4,1] + y[5,1] + y[6,1] + nse[1] = 1678.3
 demand_met[2] : y[1,2] + y[2,2] + y[3,2] + y[4,2] + y[5,2] + y[6,2] + nse[2] = 1596.7
 demand_met[3] : y[1,3] + y[2,3] + y[3,3] + y[4,3] + y[5,3] + y[6,3] + nse[3] = 1522.8
 demand_met[4] : y[1,4] + y[2,4] + y[3,4] + y[4,4] + y[5,4] + y[6,4] + nse[4] = 1497.7
 demand_met[5] : y[1,5] + y[2,5] + y[3,5] + y[4,5] + y[5,5] + y[6,5] + nse[5] = 1507.8
 demand_met[6] : y[1,6] + y[2,6] + y[3,6] + y[4,6] + y[5,6] + y[6,6] + nse[6] = 1540.2
 demand_met[7] : y[1,7] + y[2,7] + y[3,7] + y[4,7] + y[5,7] + y[6,7] + nse[7] = 1591.2
 demand_met[8] : y[1,8] + y[2,8] + y[3,8] + y[4,8] + y[5,8] + y[6,8] + nse[8] = 1657.3
 demand_met[9] : y[1,9] + y[2,9] + y[3,9] + y[4,9] + y[5,9] + y[6,9] + nse[9] = 1677.1
 demand_met[10] : y[1,10] + y[2,10] + y[3,10] + y[4,10] + y[5,10] + y[6,10] + nse[10] = 1809
 demand_met[11] : y[1,11] + y[2,11] + y[3,11] + y[4,11] + y[5,11] + y[6,11] + nse[11] = 1895
 demand_met[12] : y[1,12] + y[2,12] + y[3,12] + y[4,12] + y[5,12] + y[6,12] + nse[12] = 1944
 demand_met[13] : y[1,13] + y[2,13] + y[3,13] + y[4,13] + y[5,13] + y[6,13] + nse[13] = 1959

 demand_met[8749] : y[1,8749] + y[2,8749] + y[3,8749] + y[4,8749] + y[5,8749] + y[6,8749] + n
```

```
 demand_met[8750] : y[1,8750] + y[2,8750] + y[3,8750] + y[4,8750] + y[5,8750] + y[6,8750] + r
 demand_met[8751] : y[1,8751] + y[2,8751] + y[3,8751] + y[4,8751] + y[5,8751] + y[6,8751] + r
 demand_met[8752] : y[1,8752] + y[2,8752] + y[3,8752] + y[4,8752] + y[5,8752] + y[6,8752] + r
 demand_met[8753] : y[1,8753] + y[2,8753] + y[3,8753] + y[4,8753] + y[5,8753] + y[6,8753] + r
 demand_met[8754] : y[1,8754] + y[2,8754] + y[3,8754] + y[4,8754] + y[5,8754] + y[6,8754] + r
 demand_met[8755] : y[1,8755] + y[2,8755] + y[3,8755] + y[4,8755] + y[5,8755] + y[6,8755] + r
 demand_met[8756] : y[1,8756] + y[2,8756] + y[3,8756] + y[4,8756] + y[5,8756] + y[6,8756] + r
 demand_met[8757] : y[1,8757] + y[2,8757] + y[3,8757] + y[4,8757] + y[5,8757] + y[6,8757] + r
 demand_met[8758] : y[1,8758] + y[2,8758] + y[3,8758] + y[4,8758] + y[5,8758] + y[6,8758] + r
 demand_met[8759] : y[1,8759] + y[2,8759] + y[3,8759] + y[4,8759] + y[5,8759] + y[6,8759] + r
 demand_met[8760] : y[1,8760] + y[2,8760] + y[3,8760] + y[4,8760] + y[5,8760] + y[6,8760] + r
```

Problem 1.5 (5 points)

Find the optimal solution. How much should the utility build of each type of generating plant? What will the total cost be? How much energy will be non-served?

Solution:

First, let's find the solution.

```
optimize!(gencap);
```

```
Running HiGHS 1.5.3 [date: 1970-01-01, git hash: 45a127b78]
Copyright (c) 2023 HiGHS under MIT licence terms
Presolving model
56856 rows, 56862 cols, 153048 nonzeros
56856 rows, 56862 cols, 153048 nonzeros
Presolve : Reductions: rows 56856(-4464); columns 56862(-4464); elements 153048(-8928)
Solving the presolved LP
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0     0.0000000000e+00 Pr: 8760(4.52383e+06) 0s
      41807     6.4458519665e+08 Pr: 5(0.421305); Du: 0(8.93269e-08) 5s
      41818     6.4455540055e+08 Pr: 0(0); Du: 0(3.01981e-14) 5s
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Simplex   iterations: 41818
Objective value     :  6.4455540055e+08
HiGHS run time      :          5.40
```

We can find how much generating capacity we want to build for each plant type by querying the relevant decision variable `x`. We will turn this into a DataFrame to make the presentation easier.

```
built_cap = value.(x).data
DataFrame(Plant=gens.Plant, Capacity=round.(built_cap; digits=0))
```

	Plant	Capacity
	String15	Float64
1	Geothermal	0.0
2	Coal	0.0
3	NG CCGT	1644.0
4	NG CT	687.0
5	Wind	535.0
6	Solar	1945.0

Similarly, we can find the total amount of un-served energy by adding up `nse[t]`.

```
sum(value.(nse).data)
```

```
5966.0484237646615
```

This plan results in 6000 MWh of un-served energy throughout the year.

Finally, to get the total cost of the system we can use `objective_value`:

```
objective_value(gencap)
```

```
6.445554005499631e8
```

The total cost of this plan is $\$6.4 \times 10^8$.

Problem 1.6 (5 points)

What fraction of annual generation does each plant type produce? How does this compare to the breakdown of built capacity that you found in Problem 1.5? Do these results make sense given the generator data?

Solution:

To find the total annual generation from each plant, we want to sum up the values of the variable `y` along the time dimension.

```
annual_gen = [sum(value.(y[g, :]).data) for g in G]
```

```
6-element Vector{Float64}:
      0.0
      0.0
      8.5461980360224e6
 448851.27357040567
      1.5668370608827937e6
      5.800053481100636e6
```

We can then convert this into fractions of total generation, which we can compare to fractions of built capacity.

```
annual_gen_frac = annual_gen ./ sum(annual_gen)
built_frac = built_cap ./ sum(built_cap)
DataFrame(Plant=gens[!, :Plant], Built=built_frac, Generated=annual_gen_frac)
```

	Plant	Built	Generated
	String15	Float64	Float64
1	Geothermal	0.0	0.0
2	Coal	0.0	0.0
3	NG CCGT	0.341812	0.522322
4	NG CT	0.142835	0.0274326
5	Wind	0.111116	0.0957611
6	Solar	0.404238	0.354484

One observation is that that we have to overbuild the fraction of combustion turbine gas plants (NG CT) as these are needed when wind and solar is low, but otherwise are less commonly used. We also have to slightly overbuild wind and solar relative to the power generated by these technologies, as although they are free to generate, they can be severely constrained in terms of capacity in a given hour.

Problem 2 (18 points)

The NY state legislature is considering enacting an annual CO_2 limit, which for the utility would limit the emissions in its footprint to 1.5 $MtCO_2$/yr.

Problem 2.1 (3 points)

What changes are needed to your linear program from Problem 1? Re-formulate any part of the problem that has changed.

Solution:

The only change is to add in a constraint for the CO_2 limit. Let $emis_g$ be the CO_2 emissions factor (tCO_2/MWh generated) for plant g. Then this constraint is:

$$\sum_{g \in G} emis_g \times \sum_{t \in T} y_{g,t} \leq 1500000.$$

Problem 2.2 (3 points)

Implement the new optimization problem in `JuMP`.

Solution:

We can actually just add a new constraint to the `JuMP` model (just be careful when evaluating the notebook cells!), but if you re-formulated the model object with a new name, that works as well.

```
# add in the new constraint
@constraint(gencap, co2, sum(gens[:, :Emissions] .* [sum(y[g, :]) for g in G]) <= 1500000);
```

Problem 2.3 (5 points)

Find the optimal solution. How much should the utility build of each type of generating plant? What is different from your plan from Problem 1? Do these changes make sense?

Solution:

Find the new solution:

```
optimize!(gencap)
built_cap_co2 = value.(x).data
DataFrame(Plant=gens.Plant, Capacity=round.(built_cap_co2; digits=0))
```

```
Solving LP without presolve or with basis
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0     6.4458523850e+08 Pr: 1(605433); Du: 0(2.50664e-08) 5s
       1246     6.6076367740e+08 Pr: 3294(1.82456e+06); Du: 0(7.23809e-08) 11s
       3079     6.6495329671e+08 Pr: 1011(2.67715e+06); Du: 0(7.23809e-08) 16s
```

```
      5371     6.6495423244e+08 Pr: 1061(2.69989e+06); Du: 0(7.23809e-08) 22s
      7475     6.6495558240e+08 Pr: 1066(2.71606e+06); Du: 0(7.23809e-08) 27s
      8807     6.7291058984e+08 Pr: 4724(1.33195e+06); Du: 0(2.50664e-08) 33s
      9956     6.8217236037e+08 Pr: 9467(1.99397e+07); Du: 0(1.02824e-07) 38s
     11150     6.9287309021e+08 Pr: 2945(627493); Du: 0(1.02824e-07) 44s
     12945     6.9931790264e+08 Pr: 2465(425267); Du: 0(1.02824e-07) 50s
     14506     7.1568520106e+08 Pr: 2220(392161); Du: 0(1.02824e-07) 55s
     16226     7.3355564529e+08 Pr: 2138(470093); Du: 0(1.02824e-07) 60s
     17943     7.5157746965e+08 Pr: 1380(161795); Du: 0(7.77572e-08) 65s
     19975     7.7024312096e+08 Pr: 1111(75352.4); Du: 0(7.77572e-08) 70s
     22012     7.7360023773e+08 Pr: 0(0); Du: 0(3.2136e-10) 75s
Model   status      : Optimal
Simplex   iterations: 22012
Objective value     :  7.7360023773e+08
HiGHS run time      :         75.40
```

	Plant	Capacity
	String15	Float64
1	Geothermal	271.0
2	Coal	0.0
3	NG CCGT	1520.0
4	NG CT	453.0
5	Wind	2586.0
6	Solar	2119.0

To meet the emissions constraint, we build a reduced amount of natural gas, which creates some interesting changes in the rest of the mix. This natural gas capacity is replaced by a combination of 271 MW of geothermal (which was previously zero), an additional 2000 MW of wind capacity, and slightly more solar. These massive increases in wind are required to ensure adequate generation when solar is low, when otherwise we could rely on gas for that purpose.

```
sum(value.(nse).data)
```

```
9144.877170757745
```

We also increase the amount of non-served energy from Problem 1 by about 50% due to the reduced amount of built natural gas capacity.

Problem 2.4 (5 points)

What fraction of annual generation does each plant type produce? How does this compare to the breakdown of built capacity that you found in Problem 2.3? What are the differences between these results and your plan from Problem 1?

Solution:

We can just repeat the process from Problem 1.6.

```
annual_gen_co2 = [sum(value.(y[g, :]).data) for g in G]
annual_gen_frac_co2 = annual_gen_co2 ./ sum(annual_gen_co2)
built_frac_co2 = built_cap_co2 ./ sum(built_cap_co2)
DataFrame(Plant=gens[!, :Plant], Built_CO2=built_frac_co2, Generated_CO2=annual_gen_frac_co2
```

	Plant	Built_CO2	Generated_CO2	Built_Old	Generated_Old
	String15	Float64	Float64	Float64	Float64
1	Geothermal	0.0390032	0.0630212	0.0	0.0
2	Coal	0.0	0.0	0.0	0.0
3	NG CCGT	0.218746	0.203825	0.341812	0.522322
4	NG CT	0.0651528	0.00736234	0.142835	0.0274326
5	Wind	0.372175	0.404451	0.111116	0.0957611
6	Solar	0.304923	0.321341	0.404238	0.354484

The impact of the CO_2 constraint is similar to Problem 2.3: we rely on geothermal and wind heavily, so the total generation percentages is higher than previously (and for geothermal, higher than its built capacity percentage). Meanwhile, NG CT generation is limited to emergency shortfalls, and it represents only 1% of total generation. We also use NG CCGT a lot less, even though its built capacity isn't much lower in absolute terms.

Problem 2.5 (2 points)

What would the value to the utility be of allowing it to emit an additional 1000 tCO_2/yr? An additional 5000?

Solution:

To answer this, we want to find the shadow price of the CO_2 emissions constraint.

```
shadow_price(co2)
```

```
-183.86131591226533
```

Thus, every extra tCO_2/yr allowed would reduce the cost of generation by \$183. So an extra 1000 tCO_2/yr would correspond to a value of \$183,000, since the objective and constraints are linear. And an extra 5000 tCO_2/yr would be worth \$915,000.

References

List any external references consulted, including classmates.