BEE 4750/5750 Homework 3

Ian Shen-Costello (iys2)

2022-10-20

Problem 1

Problem 1.1

$$x_G$$

is the installed capacity (MW) of generator type G, where G includes geothermal, coal, CCGT, CT, Wind, and Solar.

$$y_{G,t}$$

is the production (MW) from generator type G, in a period t, over 24 hours. Production over the year is this value multiplied by 365.

Problem 1.2

The objective function is as follows,

$$\min Z = \sum_{G}^{6} C_G^{INV} x_G + 365 \sum_{G}^{6} \sum_{t=1}^{24} C_G^{OP} y_{G,t},$$

where C_G^{INV} is the investment cost and C_G^{OP} is the operating cost for generator G.

Problem 1.3

The constraints are as follows,

- Generators cannot produce more than installed capacity and availability.

$$y_G, t \leq x_G CF_{G,t}$$

- Operating production for every hour throughout the day must meet demand.

$$\sum_{G}^{6} y_{G,t} \geq D_t$$

where D_t is demand at each hour

- Non-negativity

$$x_G \geq 0$$
$$y_{G,t} \geq 0$$

Problem 1.4

```
using JuMP
using HiGHS

Z = Model(HiGHS.Optimizer)
G = 1:length(generators)
T = hours

@variable(Z, x[G] >= 0)
@variable(Z, y[G, T] >= 0)

@objective(Z, Min, investment_cost'*x + 365*(sum(op_cost.*y)) +
1000*365*(sum(y) - sum(demand)))

@constraint(Z, availablility[g in G, t in T], y[g,t] <= cf[g,t]*x[g])

@constraint(Z, load[t in T], sum(y[:, t]) == demand[t])
```

1-dimensional DenseAxisArrayConstraintRefModel, MathOptInterface.ConstraintIndexMathOptInter
MathOptInterface.EqualToFloat64, ScalarShape,1,... with index sets: Dimension 1, 1:24
And data, a 24-element VectorConstraintRefModel, MathOptInterface.ConstraintIndexMathOptInterfa
MathOptInterface.EqualToFloat64, ScalarShape: load[1] : y[1,1] + y[2,1] + y[3,1] + y[4,1] + y[5,1] + y[6,1] = 1517.0 load[2] : y[1,2] + y[2,2] + y[3,2] + y[4,2] + y[5,2] + y[6,2] = 1486.0 load[3] : y[1,3] + y[2,3] + y[3,3] + y[4,3] + y[5,3] + y[6,3] = 1544.0 load[4] : y[1,4] + y[2,4] + y[3,4] + y[4,4] + y[5,4] + y[6,4] = 1733.0 load[5] : y[1,5] + y[2,5] + y[3,5] + y[4,5] + y[5,5] + y[6,5] = 2058.0 load[6] : y[1,6] + y[2,6] + y[3,6] + y[4,6] + y[5,6] + y[6,6] = 2470.0 load[7] : y[1,7] + y[2,7] + y[3,7] + y[4,7] + y[5,7] + y[6,7] = 2628.0 load[8] : y[1,8] + y[2,8] + y[3,8] + y[4,8] + y[5,8] + y[6,8] = 2696.0 load[9] : y[1,9] + y[2,9] + y[3,9] + y[4,9] + y[5,9] + y[6,9] = 2653.0 load[10] : y[1,10] + y[2,10] + y[3,10] + y[4,10] + y[5,10] + y[6,10] = 2591.0 load[16] : y[1,16] + y[2,16] + y[3,16] + y[4,16] + y[5,16] + y[6,16] = 2821.0 load[17] : y[1,17] + y[2,17] + y[3,17] + y[4,17] + y[5,17] + y[6,17] = 3017.0 load[18] : y[1,18] + y[2,18] + y[3,18] + y[4,18] + y[5,18] + y[6,18] = 3074.0 load[19] : y[1,19] + y[2,19] + y[3,19] + y[4,19] + y[5,19] + y[6,19] = 2957.0 load[20] : y[1,20] + y[2,20] + y[3,20] + y[4,20] + y[5,20] + y[6,20] = 2487.0 load[21] : y[1,21] + y[2,21] + y[3,21] + y[4,21] + y[5,21] + y[6,21] = 2249.0 load[22] : y[1,22] + y[2,22] + y[3,22] + y[4,22] + y[5,22] + y[6,22] = 1933.0 load[23] : y[1,23] + y[2,23] + y[3,23] + y[4,23] + y[5,23] + y[6,23] = 1684.0 load[24] : y[1,24] + y[2,24] + y[3,24] + y[4,24] + y[5,24] + y[6,24] = 1563.0

Problem 1.5

```
julia> optimize!(Z)
Presolving model
156 rows, 138 cols, 396 nonzeros
156 rows, 138 cols, 396 nonzeros
Presolve : Reductions: rows 156(-12); columns 138(-12); elements 396(-24)
Solving the presolved LP
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0    -2.0818505000e+10 Pr: 24(60321.5) 0s
        120     9.1214221224e+08 Pr: 0(0) 0s
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Simplex   iterations: 120
Objective value     :  9.1214221224e+08
HiGHS run time      :          0.00

julia> objective_value(Z)
9.121422122418861e8

julia> print(value.(x))
1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, 1:6
And data, a 6-element Vector{Float64}:
    0.0
    0.0
 1704.2566371681414
  881.3274336283189
 1238.053097345133
 2728.9085545722714
julia> value.(y)
2-dimensional DenseAxisArray{Float64,2,...} with index sets:
    Dimension 1, 1:6
    Dimension 2, 1:24
And data, a 6×24 Matrix{Float64}:
  -0.0     -0.0    -0.0      -0.0      -0.0    …    -0.0     -0.0     -0.0
  -0.0     -0.0    -0.0      -0.0      -0.0         -0.0     -0.0     -0.0
 798.929  780.31  863.071  1386.35   1704.26      1227.31  1003.07   844.929
   0.0      0.0     0.0       0.0     180.416        0.0      0.0      0.0
 718.071  705.69  680.929   346.655   173.327      705.69   680.929  718.071
  -0.0     -0.0    -0.0      -0.0      -0.0    …    -0.0     -0.0     -0.0

julia> #Energy non-served
       print(365*sum(value.(y)) - sum(demand))
2.0761468000000004e7
```

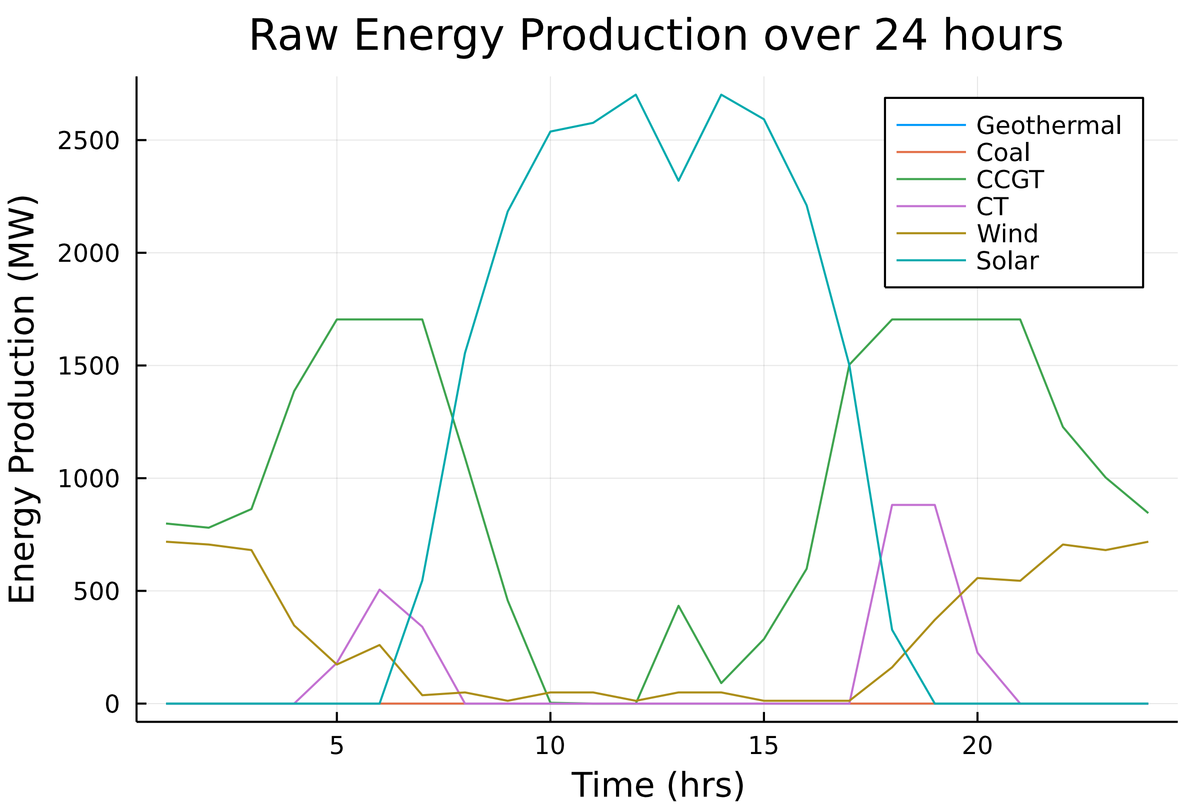
The utility build of each type of generating plant would be zero for geothermal and coal, 1704.26 MW for CCGT, 881.05 MW for CT, 1238.05 MW for wind, and 2728.91 MW for solar. (This is shown above in value.(x))

The total cost will be $912,142,212.24. (This is shown above in objective value.)

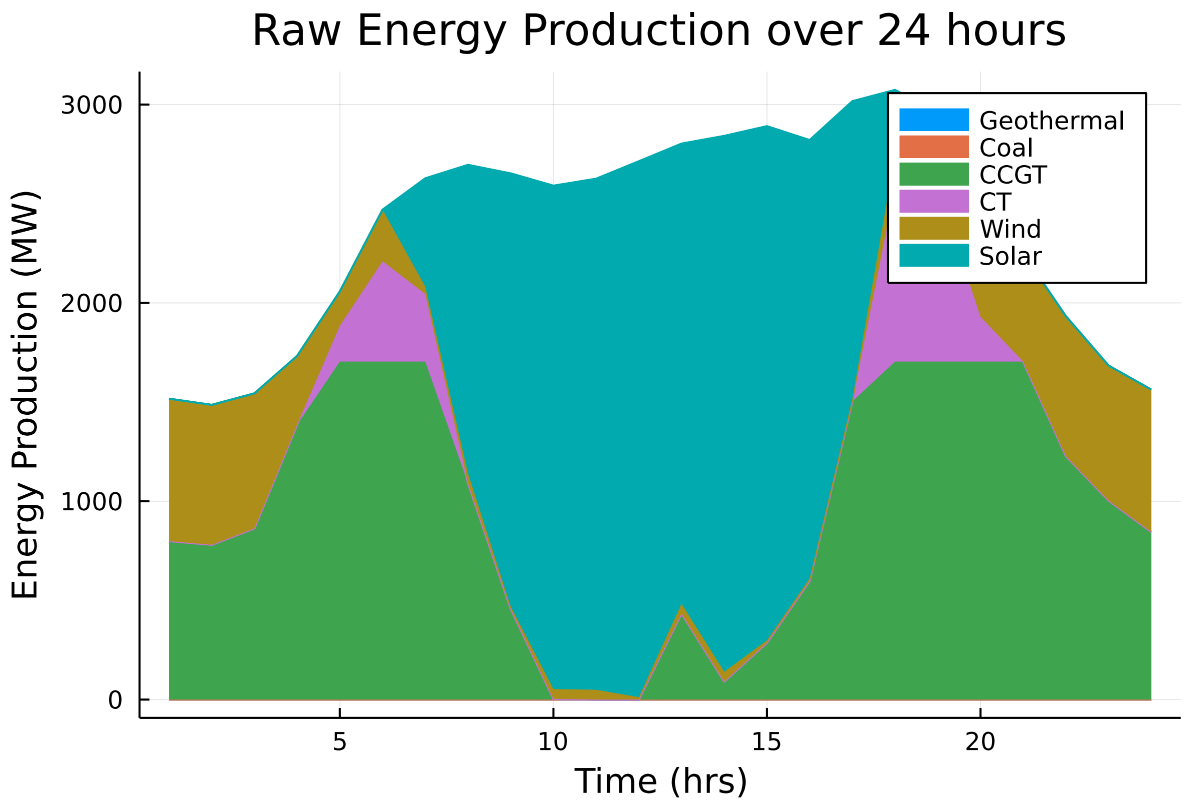
The amount of energy non-served will be 2.66 x 10^{-3} Watts.

Problem 1.6

```
julia> using Plots

julia> h = zeros(6,24)
6×24 Matrix{Float64}:
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0

julia> for i =1:6
           for j = 1:24
               h[i,j] = value.(y)[i,j]
           end
       end

julia> plot(h',title = "Raw Energy Production over 24 hours", xlabel = "Time
(hrs)", ylabel = "Energy Production (MW)", label= ["Geothermal" "Coal" "CCGT"
"CT" "Wind" "Solar"])
```




```
julia> areaplot(h',title = "Raw Energy Production over 24 hours", xlabel =
"Time (hrs)", ylabel = "Energy Production (MW)", label= ["Geothermal" "Coal"
"CCGT" "CT" "Wind" "Solar"])
```



The main takeaways are that solar production dominates during the day while the other 3 forms (CCGT, CT, and Wind) can meet demand when there is less sunlight.

Problem 2

Problem 2.1

The objective and decision variables would remain the same.

A new constraint would need to be added:

$$365 \sum_{G}^{6} \sum_{t}^{24} CO2_G \leq 1.5\text{Mt } CO_2/\text{yr}$$

Problem 2.2

```
L = Model(HiGHS.Optimizer)
G = 1:length(generators)
T = hours

@variable(L, x[G] >= 0)
@variable(L, y[G, T] >= 0)

@objective(L, Min, investment_cost'*x + 365*(sum(op_cost.*y)) +
1000*365*(sum(y) - sum(demand)))
```

```
@constraint(L, availablility[g in G, t in T], y[g,t] <= cf[g,t]*x[g])

@constraint(L, load[t in T], sum(y[:, t]) == demand[t])

@constraint(L, co2[g in G], 365*sum((co2_emissions[g]*sum(y[g,:]))) <=
1.5*10^6)
```

1-dimensional DenseAxisArrayConstraintRefModel, MathOptInterface.ConstraintIndexMathOptInte
MathOptInterface.LessThanFloat64, ScalarShape,1,... with index sets: Dimension 1, 1:6
And data, a 6-element VectorConstraintRefModel, MathOptInterface.ConstraintIndexMathOptInterfac
MathOptInterface.LessThanFloat64, ScalarShape: co2[1] : 0 1.5e6 co2[2] : 365 y[2,1] + 365 y[2,2] + 365 y[2,3] + 365 y[2,4] + 365 y[2,5] + 365 y[2,6] + 365 y[2,7] + 365 y[2,8] + 365 y[2,9] + 365 y[2,10] + 365 y[2,11] + 365 y[2,12] + 365 y[2,13] + 365 y[2,14] + 365 y[2,15] + 365 y[2,16] + 365 y[2,17] + 365 y[2,18] + 365 y[2,19] + 365 y[2,20] + 365 y[2,21] + 365 y[2,22] + 365 y[2,23] + 365 y[2,24] 1.5e6 co2[3] : 156.95 y[3,1] + 156.95 y[3,2] + 156.95 y[3,3] + 156.95 y[3,4] + 156.95 y[3,5] + 156.95 y[3,6] + 156.95 y[3,7] + 156.95 y[3,8] + 156.95 y[3,9] + 156.95 y[3,10] + 156.95 y[3,11] + 156.95 y[3,12] + 156.95 y[3,13] + 156.95 y[3,14] + 156.95 y[3,15] + 156.95 y[3,16] + 156.95 y[3,17] + 156.95 y[3,18] + 156.95 y[3,19] + 156.95 y[3,20] + 156.95 y[3,21] + 156.95 y[3,22] + 156.95 y[3,23] + 156.95 y[3,24] 1.5e6 co2[4] : 200.75000000000003 y[4,1] + 200.75000000000003 y[4,2] + 200.75000000000003 y[4,3] + 200.75000000000003 y[4,4] + 200.75000000000003 y[4,5] + 200.75000000000003 y[4,6] + 200.75000000000003 y[4,7] + 200.75000000000003 y[4,8] + 200.75000000000003 y[4,9] + 200.75000000000003 y[4,10] + 200.75000000000003 y[4,11] + 200.75000000000003 y[4,12] + 200.75000000000003 y[4,13] + 200.75000000000003 y[4,14] + 200.75000000000003 y[4,15] + 200.75000000000003 y[4,16] + 200.75000000000003 y[4,17] + 200.75000000000003 y[4,18] + 200.75000000000003 y[4,19] + 200.75000000000003 y[4,20] + 200.75000000000003 y[4,21] + 200.75000000000003 y[4,22] + 200.75000000000003 y[4,23] + 200.75000000000003 y[4,24] 1.5e6 co2[5] : 0 1.5e6 co2[6] : 0 1.5e6

Problem 2.3

```
julia> optimize!(L)
Presolving model
159 rows, 138 cols, 468 nonzeros
159 rows, 138 cols, 468 nonzeros
Presolve : Reductions: rows 159(-15); columns 138(-12); elements 468(-24)
Solving the presolved LP
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0    -2.0818505000e+10 Pr: 24(159062) 0s
        120     9.3398845328e+08 Pr: 0(0) 0s
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Simplex   iterations: 120
Objective value     :  9.3398845328e+08
HiGHS run time      :          0.00

julia> L = objective_value(L)
9.339884532803383e8

julia> print(value.(x))
1-dimensional DenseAxisArray{Float64,1,...} with index sets:
```

```
    Dimension 1, 1:6
And data, a 6-element Vector{Float64}:
    0.0
    0.0
  813.7901170184164
 1551.4733364332083
 2668.6812691819837
 3015.0665129559793
julia> value.(y)
2-dimensional DenseAxisArray{Float64,2,...} with index sets:
    Dimension 1, 1:6
    Dimension 2, 1:24
And data, a 6×24 Matrix{Float64}:
    0.0     0.0    -0.0      -0.0     ...  -0.0      -0.0      -0.0
    0.0     0.0    -0.0      -0.0         -0.0      -0.0      -0.0
    0.0     0.0    76.2253   813.79       411.852   216.225    15.1649
    0.0     0.0     0.0      171.979        0.0       0.0       0.0
 1517.0  1486.0  1467.77     747.231     1521.15   1467.77   1547.84
   -0.0     0.0    -0.0      -0.0     ...  -0.0      -0.0      -0.0
```

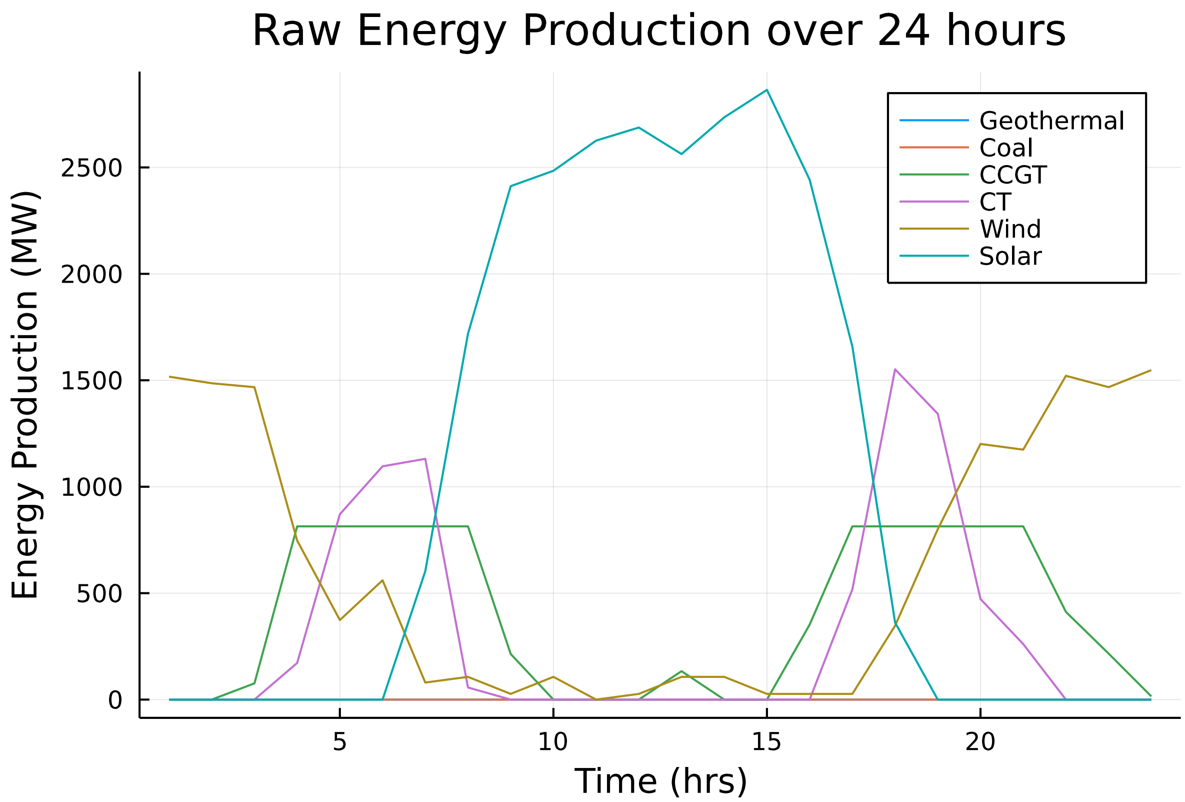
The utility build of each type of generating plant would be no purchase of geothermal and coal, 813.79 MW for CCGT, 1551.47 MW for CT, 2668.68 MW for wind, and 3015.07 MW for solar. (This is shown above in value.(x))

The main difference between this and part 1 is less investment in CCGT, as it produces a lot of carbon emissions, and more investment into wind and solar, which produce no carbon emissions.

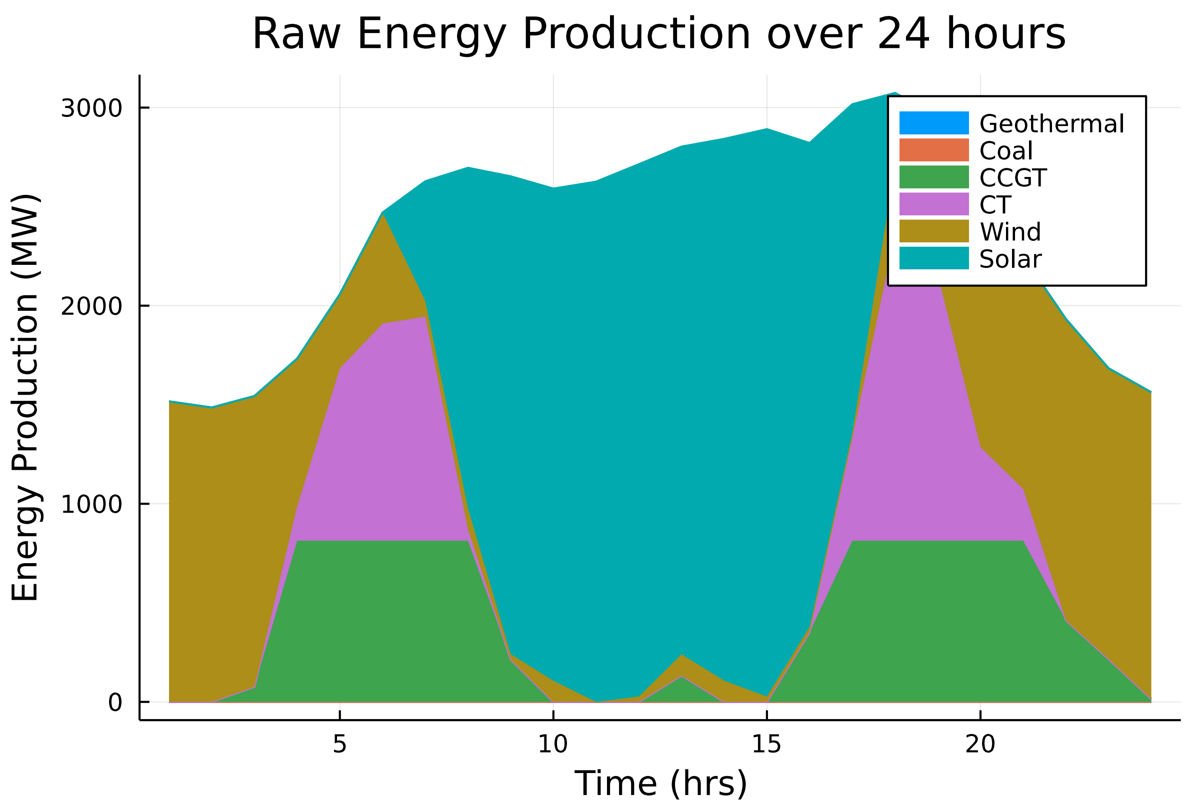
The total cost will be $933,988,453.28. The slight increase in cost makes sense as renewable energy sources, which are more expensive, are being prioritizes with the emissions limit. (This is shown above in objective value.)

Problem 2.4

```
julia> h = zeros(6,24)
6×24 Matrix{Float64}:
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0

julia> for i =1:6
           for j = 1:24
               h[i,j] = value.(y)[i,j]
           end
       end

julia> plot(h',title = "Raw Energy Production over 24 hours", xlabel = "Time
(hrs)", ylabel = "Energy Production (MW)", label= ["Geothermal" "Coal" "CCGT"
"CT" "Wind" "Solar"])
```




```
julia> areaplot(h',title = "Raw Energy Production over 24 hours", xlabel =
"Time (hrs)", ylabel = "Energy Production (MW)", label= ["Geothermal" "Coal"
"CCGT" "CT" "Wind" "Solar"])
```



The difference is a major cap or plateau in CCGT production, during peak hours. Renewables also make up a larger proportion of production overall in order to meet the carbon emissiosn requirement.

Problem 2.5

```
julia> M = Model(HiGHS.Optimizer)
A JuMP Model
Feasibility problem with:
Variables: 0
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: HiGHS

julia> G = 1:length(generators)
1:6

julia> T = hours
1:24

julia> @variable(M, x[G] >= 0)
1-dimensional DenseAxisArray{VariableRef,1,...} with index sets:
    Dimension 1, 1:6
And data, a 6-element Vector{VariableRef}:
 x[1]
 x[2]
 x[3]
 x[4]
 x[5]
 x[6]

julia> @variable(M, y[G, T] >= 0)
2-dimensional DenseAxisArray{VariableRef,2,...} with index sets:
    Dimension 1, 1:6
    Dimension 2, 1:24
And data, a 6×24 Matrix{VariableRef}:
 y[1,1]  y[1,2]  y[1,3]  y[1,4]  …  y[1,21]  y[1,22]  y[1,23]  y[1,24]
 y[2,1]  y[2,2]  y[2,3]  y[2,4]     y[2,21]  y[2,22]  y[2,23]  y[2,24]
 y[3,1]  y[3,2]  y[3,3]  y[3,4]     y[3,21]  y[3,22]  y[3,23]  y[3,24]
 y[4,1]  y[4,2]  y[4,3]  y[4,4]     y[4,21]  y[4,22]  y[4,23]  y[4,24]
 y[5,1]  y[5,2]  y[5,3]  y[5,4]     y[5,21]  y[5,22]  y[5,23]  y[5,24]
 y[6,1]  y[6,2]  y[6,3]  y[6,4]  …  y[6,21]  y[6,22]  y[6,23]  y[6,24]

julia> @objective(M, Min, investment_cost'*x + 365*(sum(op_cost.*y)) +
1000*365*(sum(y) - sum(demand)))
457000 x[1] + 268000 x[2] + 85000 x[3] + 62580 x[4] + 92000 x[5] + 92000 x[6] +
 373030 y[2,1] + 377775 y[3,1] + 381425 y[4,1] + 373030 y[2,2] + 377775 y[3,2]
+ 381425 y[4,2] + 373030 y[2,3] + 377775 y[3,3] + 381425 y[4,3] + 373030 y[2,4]
 + 377775 y[3,4] + 381425 y[4,4] + 373030 y[2,5] + 377775 y[3,5] + 381425 y
[4,5] + 373030 y[2,6] + 377775 y[3,6] + 381425 y[4,6] + 373030 y[2,7] + 377775
y[3,7] + 381425 y[4,7] + 373030 y[2,8] + 377775 y[3,8] + 381425 y[4,8] + 373030
 y[2,9] + 377775 y[3,9] + 381425 y[4,9] + 373030 y[2,10] + 377775 y[3,10] +
381425 y[4,10] + 373030 y[2,11] + 377775 y[3,11] + 381425 y[4,11] + 373030 y
[2,12] + 377775 y[3,12] + 381425 y[4,12] + 373030 y[2,13] + 377775 y[3,13] +
381425 y[4,13] + 373030 y[2,14] + 377775 y[3,14] + 381425 y[4,14] + 373030 y
[2,15] + 377775 y[3,15] + 381425 y[4,15] + 373030 y[2,16] + 377775 y[3,16] +
381425 y[4,16] + 373030 y[2,17] + 377775 y[3,17] + 381425 y[4,17] + 373030 y
[2,18] + 377775 y[3,18] + 381425 y[4,18] + 373030 y[2,19] + 377775 y[3,19] +
```

```
381425 y[4,19] + 373030 y[2,20] + 377775 y[3,20] + 381425 y[4,20] + 373030 y
[2,21] + 377775 y[3,21] + 381425 y[4,21] + 373030 y[2,22] + 377775 y[3,22] +
381425 y[4,22] + 373030 y[2,23] + 377775 y[3,23] + 381425 y[4,23] + 373030 y
[2,24] + 377775 y[3,24] + 381425 y[4,24] + 365000 y[1,1] + 365000 y[5,1] +
365000 y[6,1] + 365000 y[1,2] + 365000 y[5,2] + 365000 y[6,2] + 365000 y[1,3] +
 365000 y[5,3] + 365000 y[6,3] + 365000 y[1,4] + 365000 y[5,4] + 365000 y[6,4]
+ 365000 y[1,5] + 365000 y[5,5] + 365000 y[6,5] + 365000 y[1,6] + 365000 y[5,6]
 + 365000 y[6,6] + 365000 y[1,7] + 365000 y[5,7] + 365000 y[6,7] + 365000 y
[1,8] + 365000 y[5,8] + 365000 y[6,8] + 365000 y[1,9] + 365000 y[5,9] + 365000
y[6,9] + 365000 y[1,10] + 365000 y[5,10] + 365000 y[6,10] + 365000 y[1,11] +
365000 y[5,11] + 365000 y[6,11] + 365000 y[1,12] + 365000 y[5,12] + 365000 y
[6,12] + 365000 y[1,13] + 365000 y[5,13] + 365000 y[6,13] + 365000 y[1,14] +
365000 y[5,14] + 365000 y[6,14] + 365000 y[1,15] + 365000 y[5,15] + 365000 y
[6,15] + 365000 y[1,16] + 365000 y[5,16] + 365000 y[6,16] + 365000 y[1,17] +
365000 y[5,17] + 365000 y[6,17] + 365000 y[1,18] + 365000 y[5,18] + 365000 y
[6,18] + 365000 y[1,19] + 365000 y[5,19] + 365000 y[6,19] + 365000 y[1,20] +
365000 y[5,20] + 365000 y[6,20] + 365000 y[1,21] + 365000 y[5,21] + 365000 y
[6,21] + 365000 y[1,22] + 365000 y[5,22] + 365000 y[6,22] + 365000 y[1,23] +
365000 y[5,23] + 365000 y[6,23] + 365000 y[1,24] + 365000 y[5,24] + 365000 y
[6,24] - 20818505000

julia> @constraint(M, availablility[g in G, t in T], y[g,t] <= cf[g,t]*x[g])
2-dimensional DenseAxisArray{ConstraintRef{Model, MathOptInterface.
ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64},
MathOptInterface.LessThan{Float64}}, ScalarShape},2,...} with index sets:
    Dimension 1, 1:6
    Dimension 2, 1:24
And data, a 6×24 Matrix{ConstraintRef{Model, MathOptInterface.ConstraintIndex
{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.LessThan
{Float64}}, ScalarShape}}:
 availablility[1,1] : -0.95 x[1] + y[1,1] ≤ 0.0  …  availablility[1,24] :
-0.95 x[1] + y[1,24] ≤ 0.0
 availablility[2,1] : -x[2] + y[2,1] ≤ 0.0          availablility[2,24] : -x[2]
 + y[2,24] ≤ 0.0
 availablility[3,1] : -x[3] + y[3,1] ≤ 0.0          availablility[3,24] : -x[3]
 + y[3,24] ≤ 0.0
 availablility[4,1] : -x[4] + y[4,1] ≤ 0.0          availablility[4,24] : -x[4]
 + y[4,24] ≤ 0.0
 availablility[5,1] : -0.58 x[5] + y[5,1] ≤ 0.0     availablility[5,24] : -0.58
 x[5] + y[5,24] ≤ 0.0
 availablility[6,1] : y[6,1] ≤ 0.0               …  availablility[6,24] : y
[6,24] ≤ 0.0

julia> @constraint(M, load[t in T], sum(y[:, t]) == demand[t])
1-dimensional DenseAxisArray{ConstraintRef{Model, MathOptInterface.
ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64},
MathOptInterface.EqualTo{Float64}}, ScalarShape},1,...} with index sets:
    Dimension 1, 1:24
And data, a 24-element Vector{ConstraintRef{Model, MathOptInterface.
ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64},
MathOptInterface.EqualTo{Float64}}, ScalarShape}}:
 load[1] : y[1,1] + y[2,1] + y[3,1] + y[4,1] + y[5,1] + y[6,1] = 1517.0
 load[2] : y[1,2] + y[2,2] + y[3,2] + y[4,2] + y[5,2] + y[6,2] = 1486.0
 load[3] : y[1,3] + y[2,3] + y[3,3] + y[4,3] + y[5,3] + y[6,3] = 1544.0
 load[4] : y[1,4] + y[2,4] + y[3,4] + y[4,4] + y[5,4] + y[6,4] = 1733.0
 load[5] : y[1,5] + y[2,5] + y[3,5] + y[4,5] + y[5,5] + y[6,5] = 2058.0
 load[6] : y[1,6] + y[2,6] + y[3,6] + y[4,6] + y[5,6] + y[6,6] = 2470.0
 load[7] : y[1,7] + y[2,7] + y[3,7] + y[4,7] + y[5,7] + y[6,7] = 2628.0
```

```
 load[8] : y[1,8] + y[2,8] + y[3,8] + y[4,8] + y[5,8] + y[6,8] = 2696.0
 load[9] : y[1,9] + y[2,9] + y[3,9] + y[4,9] + y[5,9] + y[6,9] = 2653.0
 load[10] : y[1,10] + y[2,10] + y[3,10] + y[4,10] + y[5,10] + y[6,10] = 2591.0
 ⋮
 load[16] : y[1,16] + y[2,16] + y[3,16] + y[4,16] + y[5,16] + y[6,16] = 2821.0
 load[17] : y[1,17] + y[2,17] + y[3,17] + y[4,17] + y[5,17] + y[6,17] = 3017.0
 load[18] : y[1,18] + y[2,18] + y[3,18] + y[4,18] + y[5,18] + y[6,18] = 3074.0
 load[19] : y[1,19] + y[2,19] + y[3,19] + y[4,19] + y[5,19] + y[6,19] = 2957.0
 load[20] : y[1,20] + y[2,20] + y[3,20] + y[4,20] + y[5,20] + y[6,20] = 2487.0
 load[21] : y[1,21] + y[2,21] + y[3,21] + y[4,21] + y[5,21] + y[6,21] = 2249.0
 load[22] : y[1,22] + y[2,22] + y[3,22] + y[4,22] + y[5,22] + y[6,22] = 1933.0
 load[23] : y[1,23] + y[2,23] + y[3,23] + y[4,23] + y[5,23] + y[6,23] = 1684.0
 load[24] : y[1,24] + y[2,24] + y[3,24] + y[4,24] + y[5,24] + y[6,24] = 1563.0

julia> @constraint(M, co2[g in G], 365*sum((co2_emissions[g]*sum(y[g,:]))) <=
1.501*10^6)
1-dimensional DenseAxisArray{ConstraintRef{Model, MathOptInterface.
ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64},
MathOptInterface.LessThan{Float64}}, ScalarShape},1,...} with index sets:
    Dimension 1, 1:6
And data, a 6-element Vector{ConstraintRef{Model, MathOptInterface.
ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64},
MathOptInterface.LessThan{Float64}}, ScalarShape}}:
 co2[1] : 0 ≤ 1.501e6
 co2[2] : 365 y[2,1] + 365 y[2,2] + 365 y[2,3] + 365 y[2,4] + 365 y[2,5] + 365
y[2,6] + 365 y[2,7] + 365 y[2,8] + 365 y[2,9] + 365 y[2,10] + 365 y[2,11] + 365
 y[2,12] + 365 y[2,13] + 365 y[2,14] + 365 y[2,15] + 365 y[2,16] + 365 y[2,17]
+ 365 y[2,18] + 365 y[2,19] + 365 y[2,20] + 365 y[2,21] + 365 y[2,22] + 365 y
[2,23] + 365 y[2,24] ≤ 1.501e6
 co2[3] : 156.95 y[3,1] + 156.95 y[3,2] + 156.95 y[3,3] + 156.95 y[3,4] +
156.95 y[3,5] + 156.95 y[3,6] + 156.95 y[3,7] + 156.95 y[3,8] + 156.95 y[3,9] +
 156.95 y[3,10] + 156.95 y[3,11] + 156.95 y[3,12] + 156.95 y[3,13] + 156.95 y
[3,14] + 156.95 y[3,15] + 156.95 y[3,16] + 156.95 y[3,17] + 156.95 y[3,18] +
156.95 y[3,19] + 156.95 y[3,20] + 156.95 y[3,21] + 156.95 y[3,22] + 156.95 y
[3,23] + 156.95 y[3,24] ≤ 1.501e6
 co2[4] : 200.75000000000003 y[4,1] + 200.75000000000003 y[4,2] +
200.75000000000003 y[4,3] + 200.75000000000003 y[4,4] + 200.75000000000003 y
[4,5] + 200.75000000000003 y[4,6] + 200.75000000000003 y[4,7] +
200.75000000000003 y[4,8] + 200.75000000000003 y[4,9] + 200.75000000000003 y
[4,10] + 200.75000000000003 y[4,11] + 200.75000000000003 y[4,12] +
200.75000000000003 y[4,13] + 200.75000000000003 y[4,14] + 200.75000000000003 y
[4,15] + 200.75000000000003 y[4,16] + 200.75000000000003 y[4,17] +
200.75000000000003 y[4,18] + 200.75000000000003 y[4,19] + 200.75000000000003 y
[4,20] + 200.75000000000003 y[4,21] + 200.75000000000003 y[4,22] +
200.75000000000003 y[4,23] + 200.75000000000003 y[4,24] ≤ 1.501e6
 co2[5] : 0 ≤ 1.501e6
 co2[6] : 0 ≤ 1.501e6

julia> optimize!(M)
Presolving model
159 rows, 138 cols, 468 nonzeros
159 rows, 138 cols, 468 nonzeros
Presolve : Reductions: rows 159(-15); columns 138(-12); elements 468(-24)
Solving the presolved LP
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0    -2.0818505000e+10 Pr: 24(159062) 0s
```

```
        120     9.3393330482e+08 Pr: 0(0) 0s
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Simplex   iterations: 120
Objective value     :  9.3393330482e+08
HiGHS run time      :          0.00

julia> M = objective_value(M)
9.339333048215866e8

julia> print(value.(x))
1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, 1:6
And data, a 6-element Vector{Float64}:
    0.0
    0.0
  813.824069940046
 1551.7793552294854
 2666.040442264474
 3015.0943111340584
julia> value.(y)
2-dimensional DenseAxisArray{Float64,2,...} with index sets:
    Dimension 1, 1:6
    Dimension 2, 1:24
And data, a 6×24 Matrix{Float64}:
    0.0     0.0    -0.0      -0.0    ...   -0.0      -0.0      -0.0
    0.0     0.0    -0.0      -0.0         -0.0      -0.0      -0.0
    0.0     0.0    77.6778  813.824       413.357   217.678    16.6965
    0.0     0.0     0.0     172.685         0.0       0.0       0.0
 1517.0  1486.0  1466.32    746.491      1519.64   1466.32   1546.3
   -0.0     0.0    -0.0      -0.0    ...   -0.0      -0.0      -0.0

julia> print(M-L)
-55148.45875167847
```

The model was run again this time with the slight increase to the carbon emission constraint.

Allowing an extra 1000tCO2/yr decreases the optimal cost to the utility by $55,148.

References

Alex Wang Grace Lin