REAL-TIME ENERGY MARKETS

IEOR 4004 - OPTIMIZATION MODELS AND METHODS

SUBMITTED TO: PROFESSOR DANIEL BIENSTOCK

TEAM: HEATWAVE

HARSHAL POPAT, HIMANGSHU CHOWDHURY, KSHEERAJA GUTURU, KUNAL SOJATIA, SREE VENKATESH RAMIREDDI

TABLE OF CONTENTS

INTRODUCTION

Economic dispatch problem is concerned with finding how much power each unit should generate for a given demand, while minimizing the total operational costs.

FIRST TASK - FIRST COMPONENT

The objective function is to minimize the sum product of per unit cost and generation amounts to the lowest possible values. Hence in the optimal solution we need to arrive at the lowest possible values for unit cost (σ_g)and generation amounts (Γ_g).

$$Minimize \sum_g \sigma_g \Gamma_g \qquad \textit{(objective)} \quad \text{---------- (1)}$$

To solve the above problem we need a linear program (that obeys the below constraints) to optimally choose the generation amounts Γ_g and the phase angles θ_i

We are given that the power flow in the line from point *i* to *j* is equal to

$$p_b = \frac{1}{x_b}\left(\theta_i - \theta_j\right) \qquad \textit{(constraint)} \quad \text{---------- (2)}$$

where, p_b = power flow on every branch (can be positive, negative or zero)

θ_i , θ_j = phase angle at '*i*' and '*j*' respectively (can be free)

x_b = reactance at branch '*b*'

To solve objective (1), for safety reasons, we need to make sure that the absolute value of power in branch b does not exceed the branch limit (u_b)

$$|p_b| \leq u_b \qquad \textit{(constraint)} \quad \text{---------- (3)}$$

As per the power balance equation, net amount of power injected by the bus *i* into the system is equal to net power being injected into the grid at *i*

$$\sum_{b \in F(i)} p_b - \sum_{b \in F(i)} p_b = \sum_{g \in G(i)} \Gamma_g - d_i \qquad \textit{(constraint)} \text{ ---------- (4)}$$

where, $\sum_{b \in F(i)} p_b$ = power flowing 'away' from i.

$\sum_{b \in F(i)} p_b$ = power flowing 'into' i.

$\sum_{g \in G(i)} \Gamma_g$ = total generation at generators located in bus i

d_i = Load at i

Additionally we have bounds that specify,

$0 \leq \Gamma_g \leq pmax_g$, for all generators g *(constraint)* ---------- (5)

METHODOLOGY:

1. Post initializing the model and adding all the variables we initialized the dictionaries for all variables (power flowing through the branch, phase angle (θ) and power output of each generator (Γ))
2. We defined the objective function (equation 1) in the model and set the target to Minimize using ***m.setObjective(objective, GRB.MINIMIZE).***
3. Constraints (equations 2,3,4 and 5 are added) to the model.
4. Model is run & Model Status, Decision Variables and objective function values are printed.

OBSERVATIONS:

1. We observe that the model is **infeasible.**

```
Coefficient statistics:
  Matrix range     [1e+00, 2e+05]
  Objective range  [4e+02, 8e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-04, 2e+01]
Presolve removed 5335 rows and 586 columns
Presolve time: 0.01s
```

```
Solved in 0 iterations and 0.01 seconds (0.00 work units)
Infeasible model
Model status: 3
Looks like the model is not feasible
```

FIRST TASK - SECOND COMPONENT

From First Component, we see that our power balance equation cannot account for the demand that exceeds the supply into the grid at i , which ultimately terms our model infeasible. To account for this imbalance, we introduce a Shed Term S_i , into the equation. This new variable models the demand that is being 'lost' or 'shed. Thereby, the new equation is

$$\sum_{b \epsilon F(i)} p_b - \sum_{b \epsilon T(i)} p_b = \sum_{b \epsilon G(i)} \Gamma_g - (d_i - S_i) \qquad \textit{(constraint)} \text{ ---------- (6)}$$

This is the new power balance equation after moderation.

Consequently, the consequent alterations are made:

1. **Constraint Addition:** $0 \leq S_i \leq d_i$ *(constraint)* ---------- (7)
2. **Objective Function Modification:** We need to keep the shed term as low as possible. We infer that if the shed is high, our costs would be high too. Therefore, we introduce a penalty term in our objective function:

$$Minimize \sum_g \sigma_g \Gamma_g + 10^6 \sum_i S_i \qquad \textit{(objective)} \text{ ---------- (8)}$$

METHODOLOGY:

1. We initialized the model similar to that of the previous model in First Component
2. We added additional Shed variables and integrated it into our existing model:

```
shed = dict()
    for node in nodes_list:
        shed[node] = m.addVar(name = 'shed_' + str(node))
    m.update()
```

3. Then, we updated the objective function to account for the cost due to the loss:

```
cost_per_generator_list =
[get_unit_cost_of_generator(g)*generation_units[g] for g in
generators_list]
    shed_at_nodes = [shed[node] for node in nodes_list]
    objective = sum(cost_per_generator_list) + 1e6*sum(shed_at_nodes)
```

4. Finally, we modified the constraints and optimized the LP:
    a. Altered power balance equation constraint:

```
m.addConstr(sum(from_power) - sum(to_power) ==
sum([generation_units[generator_] for generator_ in generators]) -
(get_load_at_node(node) - shed[node]), name = 'power_balance_' + str(node))
```

    b. Added bounds for the shed variables:

```
for node in nodes_list:
        m.addConstr(shed[node] <= get_load_at_node(node), name =
'shed_const_' + str(node))
```

OBSERVATIONS:

1. Optimal value of Objective Function (Cost): 6.086733029e+05

```
Optimize a model with 10632 rows, 6226 columns and 19839 nonzeros
Model fingerprint: 0xa5192303
Coefficient statistics:
  Matrix range     [1e+00, 2e+05]
  Objective range  [4e+02, 1e+06]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e-04, 2e+01]
Presolve removed 9502 rows and 3679 columns
Presolve time: 0.02s
Presolved: 1130 rows, 2547 columns, 6866 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0      handle free variables                          0s
Warning: 1 variables dropped from basis
```

```
Warning: switch to quad precision
    1088    6.0867330e+05   0.000000e+00   0.000000e+00      0s

Solved in 1088 iterations and 0.12 seconds (0.09 work units)
Optimal objective  6.086733029e+05
```

2. Calculated values of power flowing through the branch, phase angle (θ), power output of each generator (Γ) and shed (S)

SECOND TASK

The model for the Second Task builds upon the Model for the second component of the first task, however with a key change.

We need to study the impact of wind variability on cost, for which instead of the maximum power generation capacity $pmax_g$ being a fixed value, of generators which generate their power from wind source, we assume that $pmax_g$ has multivariate gaussian distribution because of the wind variability.

The $pmax_g$ has mean as the fixed values as used in the first task and covariance provided as in the covariance (scaledcov) data.

Our objective is to find the cost in each iteration which will be:

$$cost = \sum_i \pi_i * (d_i - S_i)$$

We keep the remaining maximum power values of generators having their source other than wind power to be the same. This is the primary difference between this model and the previous one and then using these we produce costs for 1000 iterations with each iteration having different according to the multivariate gaussian distribution.

We performed outlier detection and removal as it hampered our identification for the most probable distribution. Further, we used the Fitter python library for distribution fitting and compared the distribution of cost to some common distributions (Normal, Cauchy, Exponential, Chi Squared and Lognormal) as well as compared with more distributions extensively.

METHODOLOGY:

1. We will have to create and run the model for 1000 iterations.
2. For each iteration we initialize the model and add all the variables we initialized the dictionaries for all variables as done in the First Task-Second Component.
3. Now we generate the multivariate gaussian $pmax_g$ using Random Number Generator(default_rng) from numpy library. We have to keep in mind that the $pmax_g$ values can not be negative.

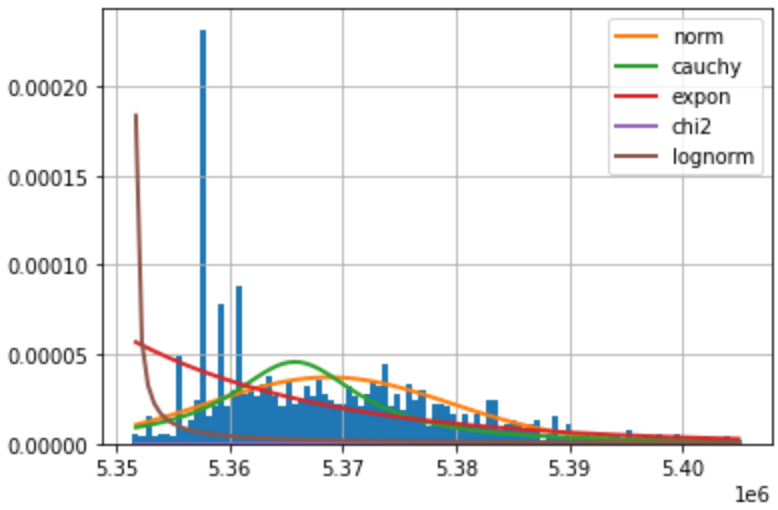
```
sample = rng.multivariate_normal(mean = np.array(wind_dataframe['pmax']),
cov = np.array(covariance_dataframe), size = None, check_valid = 'warn')
```

4. We added the rest of the constraints as defined in the First Task-Second Component.
5. We run the Model and the cost values are saved from each iteration to be written into a file for distribution analysis.
6. We use the cost data to perform distribution fitting using the Fitter package.

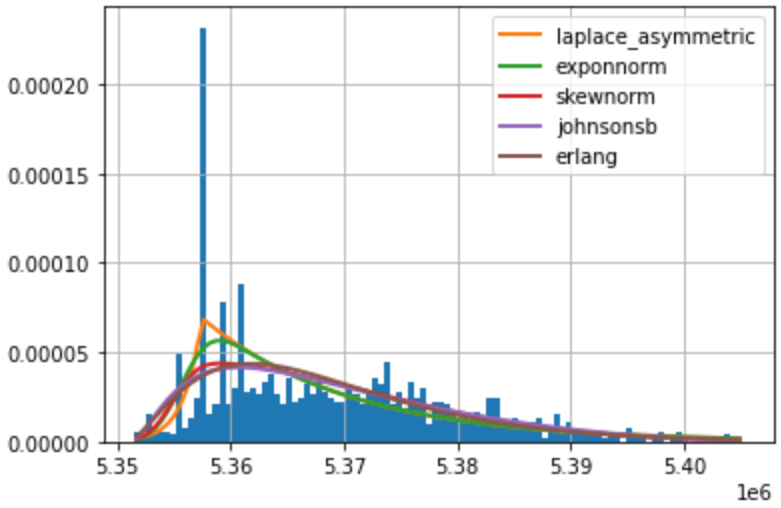
OBSERVATIONS:

1. Upon a cursory inspection of the values of the costs evaluated we can say that the distribution of the costs is left skewed.

2. When we compared the distribution of cost to some common distributions (Normal, Cauchy, Exponential, Chi Squared and Lognormal) and we found it be a most probable fit to a Normal distribution with the sum of squared error being the lowest as `5.571828e-08`.



3. Additionally, when we used 80 distributions supported by Scipy through Fitter and plotted the results and we found that Asymmetric Laplace Distribution appears to be the most probable distribution with the sum of squared error being the lowest as `4.314127e-08`.



EXTRA CREDIT

Value at Risk measures the worst expected loss over a given horizon under normal market conditions at a given level of confidence.

We are given to estimate the value-at-risk (VAR) of the public cost (defined in the report) at the 95% level. To arrive at the value-at-risk (VAR) for the costs we followed the **variance - covariance** method.

```
plt.title('VaR (Value at Risk) = ' +
str(np.percentile(np.sort(costs_dataframe['cost']), 95)))
```

The parametric method, also known as the variance-covariance method, determines the mean, or expected value, and standard deviation. The parametric method uses probability theory to calculate a maximum loss by looking at the fluctuations . In this method it is assumed that the distribution is normal, and after performing the Second Component we can say with relative certainty that the cost distribution is normal.

We used the Interior point method instead of the simplex method for this section. Interior point method has polynomial complexity in the worst case, whereas the Simplex method has a combinatorial complexity. One of the advantages of Interior Point Method is it doesn't get bogged down with degeneracy, while the simplex method may stall for many iterations at a degenerate basic solution. Using the Interior Point method we can converge to the optimal solution quickly as compared to Simplex for our application in this case.

METHODOLOGY:

1. We again initialize the model and add all the variables. We initialized the dictionaries for all variables as done in the First Task-Second Component.
2. We defined the objective function and the target.
3. Except for the generators, we added the rest of the constraints as defined in the First Task-Second Component.
4. Now for the non wind powered generators, we check whether their output is at the maximum output limit, if so we double their output limit.

```
if generator_ in non_wind_generators_reaching_max_power_list:
            m.addConstr(generation_units[generator_] <=
2*get_max_power_from_generator(generator_), name = 'max_power_' +
str(generator_))

        else:
            m.addConstr(generation_units[generator_] <=
get_max_power_from_generator(generator_), name = 'max_power_' +
str(generator_))
```

5. Now the Model is run & Model Status, Decision Variables and objective function values are printed.

OBSERVATIONS:

1. In the default case, for the Second Component, using the variance-covariance method, we observed the value-at-risk (VAR) of the public cost at the 95% level to be 5388964.49172561
2. Further, when we double the capacity of every non-wind generator whose output is at the output limit and run the Model again, we observe the the value-at-risk (VAR) of the public cost at the 95% level to be 4915042.1367675625

3. The value-at-risk (VAR) decreased by 473922.35495804716 because of the change we made.

EXTRA EXTRA CREDIT

In addition to the first task above we are given flexibility and costs associated for non wind generators (**at most 10**) as below:

a. Generation capacity is increased from $pmax_g$ to $2pmax_g$

b. Cost for this action is $\frac{\sigma_g}{10}(pmax_g)$

c. $Objective\ Function = \left\{ \begin{matrix} \sigma_g \Gamma_g, if\ \Gamma_g\ is\ at\ most\ pmax_g \\ \sigma_g pmax_g + 4\sigma_g(\Gamma_g - pmax_g)^2, if\ \Gamma_g\ is\ between\ pmax_g\ and\ 2pmax_g \end{matrix} \right\}$

Similar to the extra credit section we used the Interior Point method in this section as well. Additionally, to achieve faster computation, we set the MIPGAP parameter for Gurobi at 0.05 and the NonConvex parameter to 2 as the objective is not positive semi-definite.

METHODOLOGY:

1. We initialized the model and added the variables (phase angle, generation units and shed) similar to the first task above.
2. We added binary (0/1) for the generators for the generators whose power we can increase.

```
if generator_ not in wind_generators_array:
new_power_capacity_generators[generator_] = m.addVar(vtype = GRB.BINARY, name =
'generator_power_upgrade_' + str(generator_))
      else:
          new_power_capacity_generators[generator_] = m.addVar(vtype =
GRB.INTEGER, lb = 0, ub = 0, name = 'generator_power_upgrade_' +
str(generator_))
```

3. We added one more variable type which is multiplication of 2 decision variables (generation units and above defined indicator variable).

```
new_mixed_variable[generator_] = m.addVar(vtype = GRB.CONTINUOUS, name =
'new_mixed_variable_' + str(generator_))
   m.update()
```

```
shed = dict()
for node in nodes_list:
    shed[node] = m.addVar(name = 'shed_' + str(node))

# Integrate new variables
m.update()
```

4. We defined the objective function and constraints as per the conditions stated above and the target is set to Minimize.

OBSERVATIONS:

1. We observed the Expectation of minimum total cost (operational plus expansion cost) to be 586848.2070472378