FINAL PROJECT

IEOR E4004 OPTIMIZATION MODELS AND METHODS

REAL-TIME ENERGY MARKETS PROJECT

TEAM "THE OPTIMIZERS"

UNI	Name
xh2505	Xueying Hu
sz3019	Sitong Zhong
ar4290	Alina Rodriguez
djr2189	Daniel Rozenzaft
bd2623	Mars Dai

GOAL: *determine lowest-cost operation of a power system to satisfy demands* by solving a linear optimization problem (using a representation of power systems that ignores, simplifies or approximates several real-world features).

Modeled power system consists of a network where *buses* (i.e. nodes) are connected by *branches* (i.e. power lines) at some of the buses there are generators *g* producing power.

- Generators - each $\boldsymbol{g}$ is described by $\mathbf{pmax_g}$ – maximum output and $\boldsymbol{\sigma}_\mathbf{g}$ **-** per unit cost
- Buses - each $\boldsymbol{i}$ described by load (demand) $\boldsymbol{d}_\mathbf{i}$.
- Branches - branch $\boldsymbol{b}$ connects two buses ('from' bus and 'to' bus, positive power can flow in either direction) - described by $\boldsymbol{x_b}$ (reactance) and $\boldsymbol{u_b}$ (branch limit). ***Important:*** 'x' does not mean a variable, there may be several lines that are parallel (have the same 'from' and 'to' bus as each other).

The behavior of a power system is fully described once we specify:

- $\boldsymbol{\Gamma}_\mathbf{g}$ - the power output of each generator.
- $\boldsymbol{\Theta_i}$ ('phase angle') - for each bus *i*.

The power flow on every branch (branch b with 'from' bus i and 'to' bus j) (can be positive, negative or zero)

$$p_b = \frac{1}{x_b}(\theta_i - \theta_j) \tag{1}$$

Limit on the power flow (for safety reasons):

$$\left|p_b\right| \leq u_b \tag{2}$$

F(i) - set of branches whose 'from' bus is i,
T(i) - set of branches whose 'to' bus is i,
G(i) - set of generators located in bus i

Power balance equation at i:

$$\sum_{b\in F(i)} p_b - \sum_{b\in T(i)} p_b = \sum_{g\in G(i)} \Gamma_g - d_i \tag{3}$$

where $\sum_{b\in F(i)} p_b - \sum_{b\in T(i)} p_b$ - the load at *i* ($\sum_{b\in F(i)} p_b$ - power flowing 'away' from *i*, $\sum_{b\in T(i)} p_b$ power fl 'into' *i*); $\sum_{g\in G(i)} \Gamma_g - d_i$ - total generation at generators located in bus *i* (amount of power injected by the bus *i* into the system)

Network: 1814 buses, 2023 branches, 395 generators.

Economic dispatch

- linear program that optimally chooses the generation amounts Γ_g and the phase angles θ_i. The objective is to minimize total generating cost:

$$Minimize \sum_{g \in G(i)} \sigma_g \Gamma_g \tag{4}$$

$\boldsymbol{\Gamma_g}$ - the power output of each generator; $\boldsymbol{\sigma_g}$ - per unit cost

The constraints are (1) and (2) (for every branch b) and (3) (at every bus *i*). Additionally, we have the bounds

$$0 \leq \Gamma_g \leq pmax_g \tag{5}$$

θ_i - free (can be positive or negative) and unbounded.

First enhancement

We will consider scenarios where data changes (wind variability) cause the problem to become infeasible. To handle that, constraint (3) is modified:

$$\sum_{b \in F(i)} p_b - \sum_{b \in T(i)} p_b = \sum_{g \in G(i)} \Gamma_g - (d_i - S_i) \tag{6}$$

$S_i \in [0;\ d_i]$ - models demand that is 'lost' or 'shed' - should be kept as small as possible. (7)

We add to the objective (4) a penalty term:

$$Minimize \sum_{g \in G(i)} \sigma_g \Gamma_g + 10^6 \sum_i S_i \tag{8}$$

<u>Task 1: Formulate and solve LP</u>

Linear Program:

$$Minimize \sum_{g \in G(i)} \sigma_g \Gamma_g + 10^6 \sum_i S_i$$

s.t. $p_b = \frac{1}{x_b}(\theta_i - \theta_j)$

$$\left| p_b \right| \le u_b$$

$$0 \le \Gamma_g \le pmax_g$$

$$0 \le S_i \le d_i$$

$$\sum_{b \in F(i)} p_b - \sum_{b \in T(i)} p_b = \sum_{g \in G(i)} \Gamma_g - (d_i - S_i)$$

Code:
Relevant code files are main.py and task1.py. To gain the result, run <u>**python3 main.py 1**</u> from the top-level directory of the cloned GitHub repo in the terminal.

The LP solving driver calls the **task1** module's **setup_model** function, which begins by loading each of the datasets as a dictionary with their respective objects (generators, buses, and branches) as values. These objects are simple custom classes that enable easy access to each property and include key class methods, such as the **generators** method in the **bus** class, which returns the generators *G*(*i*) located in the bus. The **bus.branches** method similarly returns a tuple containing the *F*(*i*) and *T*(*i*) sets - these facilitate a clean implementation of Constraint *(6)*. The Gurobi model is built out using the same variable names as given in the problem description, incorporating custom class attributes and variables (as well as Gurobi's variable bounds) to set the model variables and constraints. The **setup_model** method is only called once in the package: in **main.solve**, which calls **gurobipy.model.optimize** and writes relevant information to files in **tasks/solutions**.

Solution:
In the file <u>**tasks/solutions/task1.txt**</u>, we find an optimal objective value at 608673.3029305153, and each value of θ_i, p_b, Γ_g, and S_i. These are consistent across all task1 runs.

Task 2: Pricing of demand

Objective is to study the way grid economics works - how prices are extracted from optimization problem (task1) and charged to customers. Public pays for electricity:

$$cost = \sum_i \pi_i^* (d_i - S_i) \tag{9}$$

where the sum is over all the buses, and for each bus i,

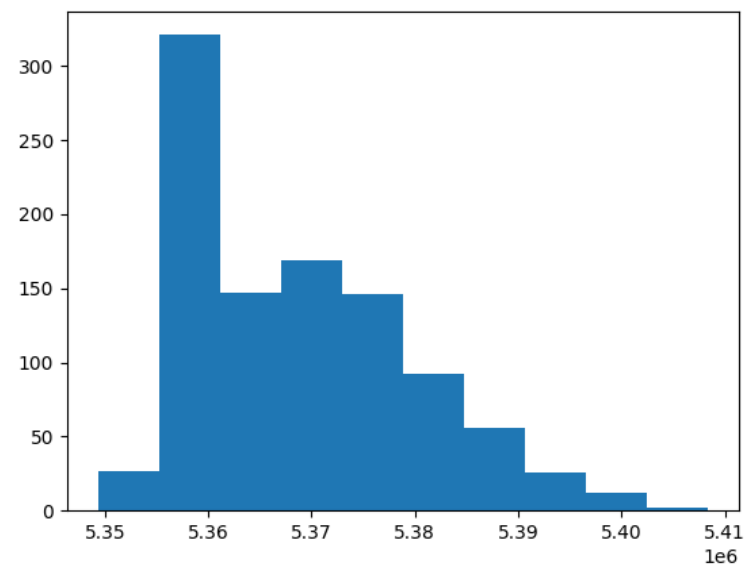
$$\pi_i^* = \; - \, Optimal\ dual\ variable\ for\ constraint\ (6).$$

In task 2, we studied the impact on cost of wind variability by following steps:
1. For $pmax$ of wind generators, we generate a multivariate Gaussian distribution with a mean of corresponding original $pmax$ and variance matrix from 'scaledcov.txt'.
2. Sample $pmax$ of the wind generators and adjust the entire $pmax$ matrix accordingly. Solved the resulting LP, then computed the cost (9).
3. Repeat step 2 1000 times, and then observe the distribution of the cost.

Linear Program:
Similar to task 1, but has updated according to the distribution sampling.

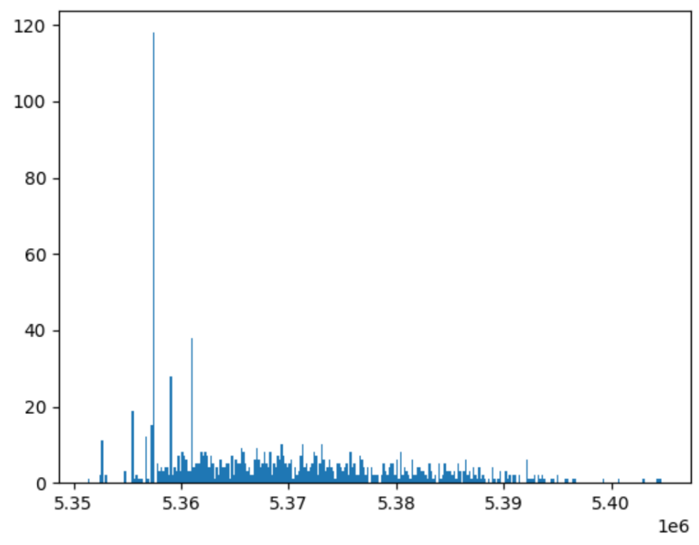
Code:
Relevant scripts are **main.py**, **tasks/task1.py** and **tasks/task2.py**. To generate the histogram located at **tasks/solutions/costs_histogram.png**, run ***python3 main.py 2*** from the top-level project directory. Helper methods in the **task2** module read the covariance matrix, call the multivariate normal distribution object provided by **NumPy** to generate random pmax values, and compute the cost values given the optimized Gurobi model.

Solution:
In the file **tasks/solutions/task2.txt**, we write the solution for the first of 1000 LP runs. In the file **tasks/solutions/task2_costs.txt**, we have the full list of calculated cost values for all 1000 samples. In the graph **tasks/solutions/task2_cost_histogram.png**, we have the distribution of 1000 costs. Over 99 percent of cost values lie between 5,350,000 and 5,541,000. We filter out outliers - costs lower than 3,600,000 - which appear with a probability of ~0.5%.



Graph 1. distribution of 1000 costs in **tasks/solutions/task2_costs_histogram.png** with 10 bins

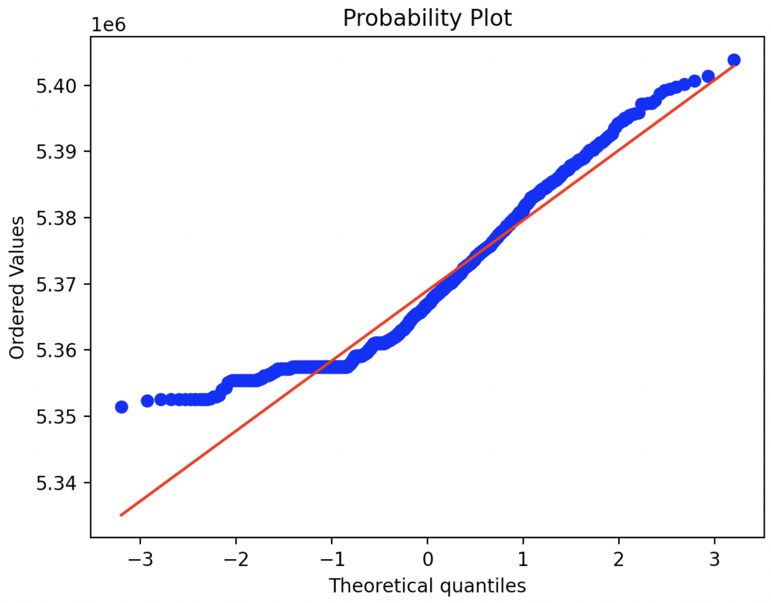
We remove a few (~0.5%) outliers for visual clarity.



Graph 2. Detailed visualization of costs generated by changing histogram's bins from 10 to 300

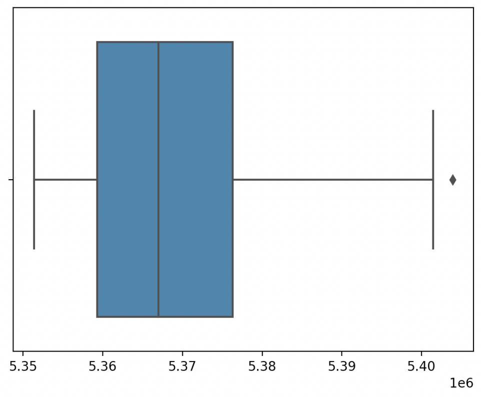
Distribution Analysis:

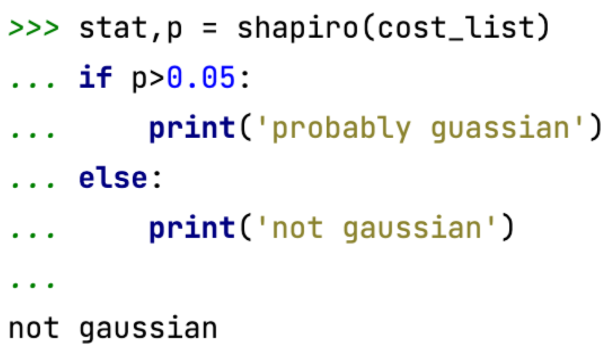
1. Normal distribution fitting



Graph 3. fitting curve for normal distribution

In graph 3, we utilized 998 data points that are larger than 5,000,000. **If data follows a normal distribution, all data points should be on the straight line that stands for the normal distribution.** However, it shows obvious deviation.





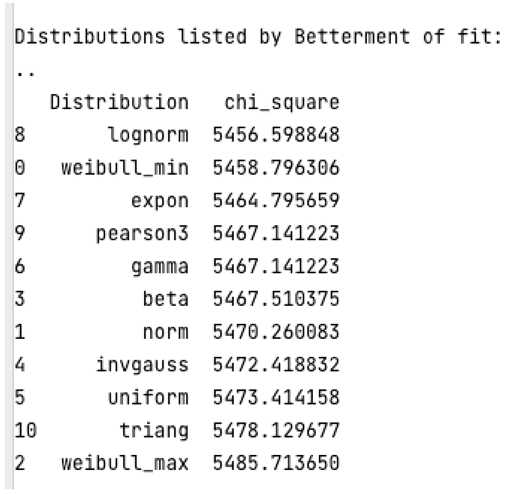
```
>>> stat,p = shapiro(cost_list)
... if p>0.05:
...     print('probably guassian')
... else:
...     print('not gaussian')
...
not gaussian
```

Graph 4. Box Plot test and Shapiro-Wilk hypothesis test for 998 sampling data

To further confirm if the cost distribution is a normal distribution, we have used the box plot test and Shapiro-Wilk hypothesis test. For the box plot test, **if the cost distribution is Gaussian, we should see the mean and median in the center of the plot.** Also for the Shapiro-Wilk hypothesis test, if the cost sampling data is gaussian, the p-value should be bigger than 5%. The result of both tests showed that this cost distribution is not Gaussian.

2. Distribution discovery

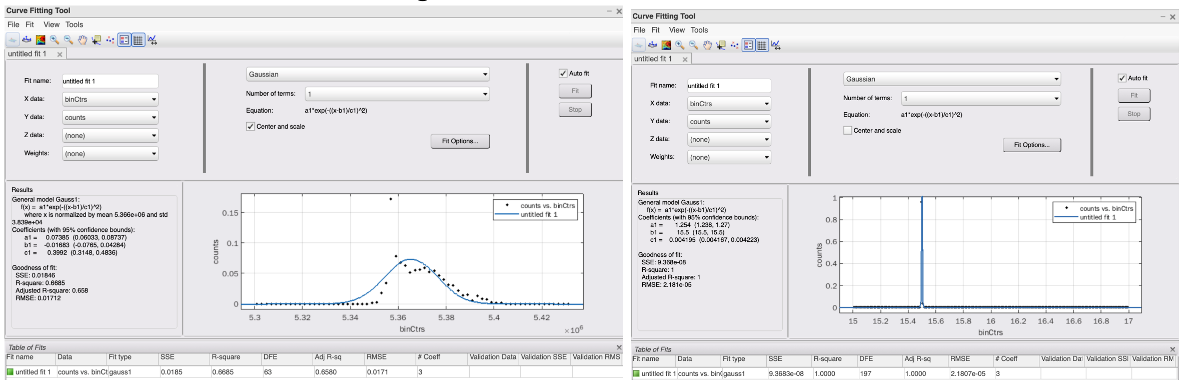
To see what distribution does the cost distribution follow, we have compared the following distribution to the cost distribution and **done the chi-square goodness-of-fit test**: (1) weibull_min (2) norm distribution (3) weibull_max (4) beta distribution (5) inv gauss distribution (6) uniform distribution (7) gamma distribution (8) exponential distribution (9) lognormal distribution (10) pearson distribution (11) triangular distribution.



```
Distributions listed by Betterment of fit:
..
   Distribution   chi_square
8       lognorm  5456.598848
0   weibull_min  5458.796306
7         expon  5464.795659
9      pearson3  5467.141223
6         gamma  5467.141223
3          beta  5467.510375
1          norm  5470.260083
4      invgauss  5472.418832
5       uniform  5473.414158
10       triang  5478.129677
2   weibull_max  5485.713650
```

Graph 5. The result of the chi-square goodness-of-fit test with **best fit being lognormal distribution**

To see how well the cost distribution is following lognormal distribution, we utilized the curve fitting tools in Matlab.
We can see that while the sum of squared errors (SSE) for normal distribution fitting is 0.01845, the SSE for lognormal distribution fitting is 9.368e-08, which is very close to 0. It is safe to say the cost distribution is close to lognormal distribution.



Curve Fitting Tool

Fit name: untitled fit 1
X data: binCtrs
Y data: counts
Z data: (none)
Weights: (none)
Gaussian
Number of terms: 1
Equation: a1*exp(-((x-b1)/c1)^2)
[✓] Center and scale
[✓] Auto fit

```
Results
General model Gauss1:
    f(x) =  a1*exp(-((x-b1)/c1)^2)
      where x is normalized by mean 5.366e+06 and std
3.839e+04
Coefficients (with 95% confidence bounds):
      a1 =     0.07385  (0.06033, 0.08737)
      b1 =    -0.01683  (-0.0765, 0.04284)
      c1 =      0.3992  (0.3148, 0.4836)

Goodness of fit:
  SSE: 0.01846
  R-square: 0.6685
  Adjusted R-square: 0.658
  RMSE: 0.01712
```

Table of Fits

Fit name	Data	Fit type	SSE	R-square	DFE	Adj R-sq	RMSE	# Coeff	Validation Data	Validation SSE	Validation RMS
🟩 untitled fit 1	counts vs. binCt	gauss1	0.0185	0.6685	63	0.6580	0.0171	3			

Curve Fitting Tool

Fit name: untitled fit 1
X data: binCtrs
Y data: counts
Z data: (none)
Weights: (none)
Gaussian
Number of terms: 1
Equation: a1*exp(-((x-b1)/c1)^2)
[ ] Center and scale
[✓] Auto fit

```
Results
General model Gauss1:
    f(x) =  a1*exp(-((x-b1)/c1)^2)
Coefficients (with 95% confidence bounds):
      a1 =       1.254  (1.238, 1.27)
      b1 =        15.5  (15.5, 15.5)
      c1 =    0.004195  (0.004167, 0.004223)

Goodness of fit:
  SSE: 9.368e-08
  R-square: 1
  Adjusted R-square: 1
  RMSE: 2.181e-05
```

Table of Fits

Fit name	Data	Fit type	SSE	R-square	DFE	Adj R-sq	RMSE	# Coeff	Validation Dat	Validation SSI	Validation RM
🟩 untitled fit 1	counts vs. bin(	gauss1	9.3683e-08	1.0000	197	1.0000	2.1807e-05	3			

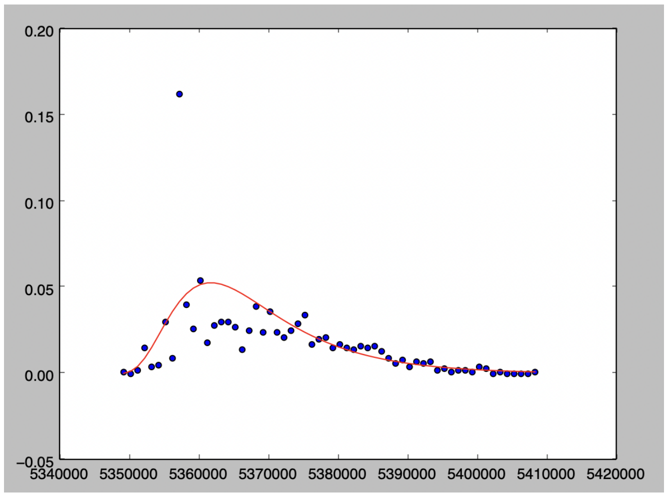
Graph 6. the curve fitting result for normal distribution and lognormal distribution

3. Lognormal distribution fitting

To find the distribution parameters, we utilized scipy.stats in Python.

Code:
Relevant script is **cost_distribution.py**. To gain the result, run **python3 cost_distribution.py 2** from the main directory.

Result:
After running the code, the parameters will be printed in the terminal and fitting curve will be generated in **tasks/solutions/task2_cost_distribution.png**.



Blue: scatter plot of costs, bins=60
Red: lognormal distribution pdf
Graph 7. cost distribution and fitted lognormal distribution in **tasks/solutions/task2_costs_distribution.png**

The **lognormal distribution parameters** are: shape = 0.4968786, loc = 5346156.917, scale = 20028.955.

Extra credit

Part A

Estimate the value-at-risk at the α=95 % level (the public cost - random variable under the distribution from 'task2').

VaR indicates maximum loss of value that may be incurred by an investor in a given period of time at a given confidence level α. In our scenario, the VAR with 95% confidence means that: **With 95% confidence, we expect that the cost will not exceed it.**

Code:
Relevant script is **cost_distribution.py**. To gain the result, run **python3 cost_distribution.py 2** from the main directory.

Result:
With the fitted lognormal distribution, we have our VAR at **5391509.70**, which is printed in the terminal.

Part B

Investigate the possibility of public cost decrease by increasing the capacity of most efficient generators, double the capacity of every non-wind generator whose output is at the output limit ($pmax_g$) as originally stated before part (a). By how much VAR decreases?

Code:
Relevant scripts are **main.py** and **cost_distribution.py**. We have added a function in main.py called **extract_capped_generators()** to capture non-wind generators who have reached their capacity. To gain the result, run **python3 main.py 3** and **python3 cost_distribution.py 3** from the main directory.

Result:
In the file **tasks/solutions/task2_costs.txt**, we have the full list of calculated cost values for all 1000 samples.
In the terminal of running **python3 cost_distribution.py 3**, we have our VAR with 95% confidence at **4921544.265390812**. Compared with part (a), we have **a VAR decrease of 8.72%**.

Part C

Update objective (8) and constraints (1), (2), (5), (6) and (7) from ‘task1’:

(1) for any non-wind generator g increase generation capacity from $pmax_g$ to $2pmax_g$ (at most for 10 generators).

(2) the cost of this action: $cost = \sigma_g \, pmax_g / 10$.

(3) if the power output (Γ_g) of the generator is at most $pmax_g$ the cost is as before: $\sigma_g\Gamma_g$, if between $pmax_g$ and $2pmax_g$ the cost is $\sigma_g pmax_g + 4\sigma_g(\Gamma_g\text{-}pmax_g)^2$.

To choose which generators to expand:

1. Calculate the expansion cost
2. Compute the expectation of the minimum operational cost (8) (similar ‘task1’).
3. The total pay is this expectation plus the expansion cost.

Note: may need to repeatedly solve LPs or quadratic optimization problems.

Code:
Relevant code files are **main.py** and **extraextracredit.py**. To gain the result, run **python3 main.py eec** in the terminal.

Solution:
Based on the lp with added binary variables, the best generators to expand are generators 0,5,7,8,13,26,99 and 340. The expectation of the cost is 1242693.89469952 when we sample from the gaussian distribution to simulate wind variability.

References:

https://www.itl.nist.gov/div898/handbook/eda/section3/eda35f.htm