User Manual for CoVEMDA Toolbox

Matlab Version 1.0

October 2021

Contents

Website

https://github.com/tamu-engineering-research/COVID-EMDA

The above website refers to a Github repository that releases the COVID-EMDA+ data hub and CoV-EMDA toolbox. Here, COVID-EMDA+ is the abbreviation for "Coronavirus Disease – Electricity Market Data Aggregation+", and CoVEMDA for "CoronaVirus – Electricity Market Data Analyzer".

Note that the Github repository provides a variety of useful resources, including the data release, the source codes, some powerful parsers, and some supplementary materials.

1 Introduction

1.1 Background

The COVID-19 pandemic is an emerging respiratory disease with high contagiousness and long incubation period [1]. Over a year has passed, it still keeps tremendous pressure on the public health systems as well as the global economy. A few early publications have also found discernible changes in the operation of power system during COVID-19 [2–4]. This is the initial motivation for an international research team from Tsinghua University and Texas A&M University to develop a powerful data hub and toolbox to track the pandemic's impacts on the power sector.

One unique feature of our work is the public and free availability. This opens up a series of opportunities for diverse stakeholders to share opinions and learn through collaboration. Another unique feature is the cross-domain resources (data, methods, and toolbox) which could provide new insights into policy decisions [5]. In some aspect, the pandemic has created a special but informative situation to assess how power systems may respond to different human interventions [6, 7]. This is certainly a positive step forward for understanding the power system resilience.

Here, we establish a data hub (COVID-EMDA+) and a supporting toolbox (CoVEMDA) on our Github repository. A variety of practical functions are tailored in the toolbox to handle the applications of baseline estimation, regression analysis, and scientific visualization. Since the world is confronting a uncertain future induced by COVID-19, this work may hopefully advance our understanding of the ongoing situations, and guide the preparations through these difficult times.

1.2 Features

We summarize the main features of our work as follows:

- Data Resources: CoVEMDA natively supports reading data, online or offline (through data archive), from the COVID-EMDA+ data hub. The released data involve electricity market data, public health data, meteorological data, and mobility data for all the existing U.S. electricity marketplaces and some typical cities[1] in these markets. Strict quality control is conducted, and external data sources are also acceptable[2].

- Baseline Estimation: Baselines are critical in quantifying the impacts of COVID-19 by estimating the counterfactual situations (We assume all influencing factors except the pandemic are happening). A few classical and representative estimation methods are collected in CoVEMDA. Rigorous comparison between baselines are also allowed.

- Regression Analysis: Regression is widely used to estimate the effect of some explanatory variables on the dependent variable. This technique helps to explore and investigate the underlying impacts of COVID-19, and two popular regression techniques—ordinary least squares regression (OLS) and vector autoregression (VAR)—are developed in CoVEMDA with flexible extensions.

- Scientific Visualization: Visualization conveys intuitive messages to readers, and becomes extremely important in explaining the pandemic-related findings. A variety of visualization designs are implemented in CoVEMDA for different applications, including graphical and statistical outputs. Simplified but extensible syntax is applied to make it handy even for beginners.

[1]Specifically, Los Angeles, Boston, New York City, Philadelphia, Chicago, Kansas City, and Houston.
[2]This feature might be helpful for user-defined extensions.

1.3 Support Team

This project is a joint effort of the group members under the supervision of Prof. Le Xie at Texas A&M University and Prof. Haiwang Zhong at Tsinghua University. The support team keeps processing, correcting, and updating the data routinely.

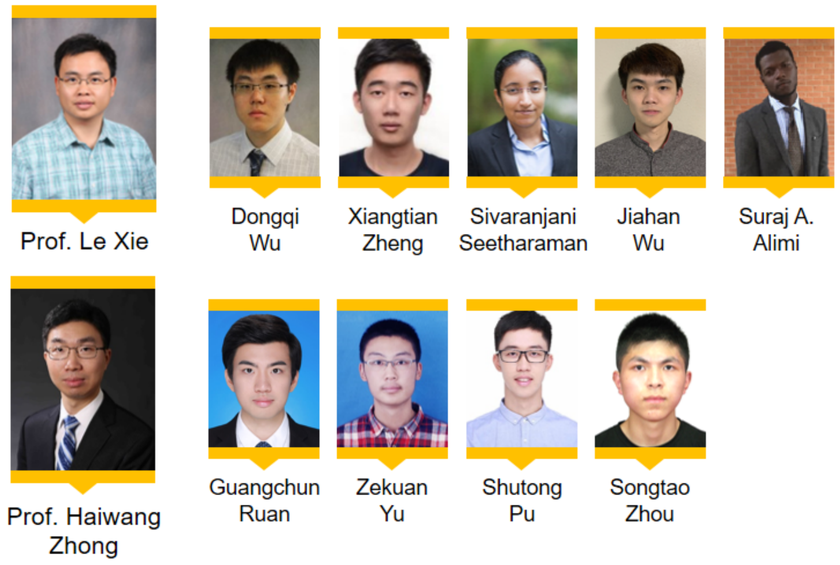

Figure 1-1: Support Team

1.4 Suggested Citation

Publications that use the data resources, supporting toolbox, or supplementary materials should consider citing the following paper accordingly. A short introduction of each paper is given as follows:

- G. Ruan, D. Wu, X. Zheng, H. Zhong, C. Kang, M. A. Dahleh, S. Sivaranjani, and L. Xie, "A cross-domain approach to analyzing the short-run impact of COVID-19 on the U.S. electricity sector," *Joule*, 2020, 4(11), 2322-2337.

  This paper gives a general introduction of the data hub, and conducts a few cross-domain analysis on the eletricity consumption across the U.S. The full text is available at: Joule and arXiv.

- G. Ruan, J. Wu, H. Zhong, Q. Xia, and L. Xie, "Quantitative assessment of U.S. bulk power systems and market operations during the COVID-19 pandemic," *Applied Energy*, 2021, 286: 116354.

  This paper substantiates the pandemic's impacts from the perspectives of power system security, electric power generation, electric power demand and electricity prices. The full text is available at: Applied Energy and arXiv.

- H. Zhong, Z. Tan, Y. He, L. Xie, and C. Kang, "Implications of COVID-19 for the electricity industry: A comprehensive review," *CSEE Journal of Power and Energy Systems*, 2020, 6(3): 489-495.

  This paper provides an early review of the global impacts that COVID-19 has caused on the electricity industry. The full text is available at: JPES.

- G. Ruan, Z. Yu, S. Pu, S. Zhou, H. Zhong, L. Xie, Q. Xia, and C. Kang, "Open-access data and toolbox for power sector analysis during COVID-19." (Preprint coming soon)

  This paper comprehensively introduces the upgrade of our data hub and toolbox. All technical details and extension settings are provided, and three in-depth empirical studies are given as well.

1.5 License

The code of CoVEMDA is licensed under the **MIT License**, and the full text can be found at https://github.com/tamu-engineering-research/COVID-EMDA/blob/master/LICENSE, shown as follows.

2 Getting Started

2.1 System Requirements

To use CoVEMDA Toolbox you will need:

- MATLAB® version 9.8 (R2020a)
- Deep Learning Toolbox™ version 14.0
- Statistics and Machine Learning Toolbox™ version 11.7
- Financial Toolbox™ version 5.15
- Econometrics Toolbox™ version 5.5

As for the hardware requirements, please refer to the System Requirements of MATLAB according to the specific version[3].

2.2 Data and Toolbox Retrieving

COVID-EMDA+ data hub and CoVEMDA toolbox are available on the Github repository. You can either download the whole package of a specific distribution or subscribe the daily-updated data via cloning the repository to your local folder.

2.2.1 Download Zip

To retrieve CoVEMDA, please download the compressed zip file from the Github repository website.

- Go to CoVEMDA Github repository.
- Click the green "Code" button.
- Click the "Download ZIP" button.
- Uncompress the zip file to a certain directory.

2.2.2 Clone Github repository

Among all methods of cloning, the following two methods are recommended for ease of use:

- Using the command-line[4].

```
git clone https://github.com/tamu-engineering-research/COVID-EMDA.git
```

- Using the Github Desktop[5].
  - Go to CoVEMDA Github repository.
  - Click the green "Code" button.
  - Click the "Open With Github Desktop" button.

[3] https://www.mathworks.com/support/requirements/previous-releases.html

[4] See https://www.git-scm.com/downloads and https://www.git-scm.com/doc for more details on Git Client.

[5] See https://desktop.github.com and https://docs.github.com/en/desktop for more details on Github Desktop

2.3 Installation

All the source codes and pretrained models in CoVEMDA toolbox are available on Github. Please get a copy of Github repository, and the toolbox is compressed and located under folder `COVID-EMDA/toolbox/`. After downloading source file, you may continue the installation as follows:

- Select a root directory and unzip the source file. Path `MATLAB/<MATLAB_version>/toolbox/` is recommended but any other directory is acceptable.
- Change the MATLAB current working path to the root directory.
- Run the installation script `install.m`.

The script `install.m` checks MATLAB and toolbox versions for CoVEMDA. If all version requirements are met, it adds all folders and subfolders of CoVEMDA source file to the MATLAB execution path.

If you need to uninstall CoVEMDA, run `uninstall.m`. It removes all folders and subfolders concerning CoVEMDA from MATLAB execution path.

2.4 Illustrative Examples

2.4.1 Baseline Estimation with an Ensemble Backcast Model

CoVEMDA provides an efficient tool for load baseline estimation: ensemble backcast model. The corresponding functions `cal_baseline` and `plot_baseline` enable the user to calculate and visualize the electricity consumption level without the impact of the pandemic. See Section 4 for more information.

For example, to calculate the baseline of NYISO, the user may enter: (command output omitted)

Command Window

```
>> t=cal_baseline('nyiso')

t =

  5x2 table

    quantile        result
    --------    --------------

       0.1      {151x25 table}
      0.25      {151x25 table}
       0.5      {151x25 table}
      0.75      {151x25 table}
       0.9      {151x25 table}
```

Access data for q=0.5:

Command Window

```
>> t.result{3}

ans =

  151x25 table

        date        00:00    01:00    02:00     ...     22:00    23:00
    ----------      -----    -----    -----     ...     -----    -----

    2020-02-01      15416    14908    14592     ...     16605    15727
    2020-02-02      15304    14793    14456     ...     16608    15735
    ...
fx  2020-06-30      16119    15376    14959     ...     18524    17146
```

If the user needs to visualize the baseline: (command output omitted)

Command Window

```
fx >> plot_baseline('nyiso','Method','backcast');
```

The result is displayed in Figure 2-1.

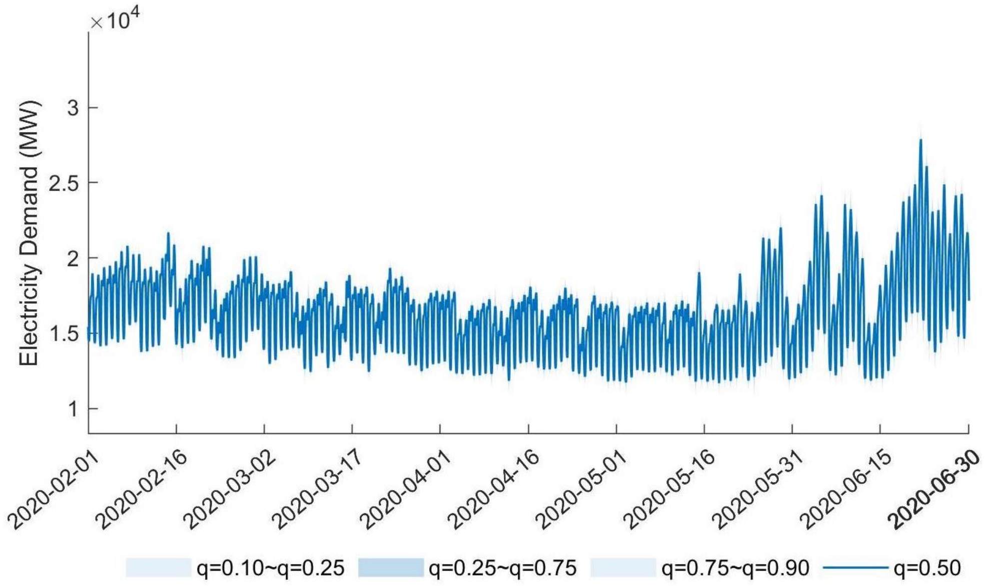

Figure 2-1: Example: Baseline and Observed Demand of NYISO

2.4.2 Regression Analysis

CoVEMDA is also capable of regression analysis. The corresponding function `OLS` and `VAR` provide regression models in which users may acquire correlation figures between variables. See Section 5 for more information.

For example, to apply a linear OLS model to the locational marginal price data of CAISO, the user may first enter the following command to retrieve data.

Command Window

```
>> t=basic_read('caiso_rto_lmp')

t =

  1462x25 table

        date        00:00     01:00     02:00                 ...
     ----------     ------    ------    ------                ...

     2017-01-19      21.53     21.69     20.86                ...
     2017-01-20      26.46     24.67     23.68                ...
     2017-01-21      30.83     28.37     28.19                ...
     2017-01-22      27.33     26.5      25.14                ...
fx                ...                 ...                     ...
```

Then some transformations are necessary in the aim of data pre-processing.

Command Window

```
>> t=basic_exchange_columns(t,1,4)

t =

  1462x25 table

     02:00     00:00     01:00         date                   ...
     ------    ------    ------    ----------                 ...

     20.86     21.53     21.69     2017-01-19                 ...
     23.68     26.46     24.67     2017-01-20                 ...
     28.19     30.83     28.37     2017-01-21                 ...
     25.14     27.33     26.5      2017-01-22                 ...
fx            ...                 ...                         ...
```

Ultimately, apply the OLS model.

Command Window

```
>> b=OLS(t,3,'linear')
               OLS Regerssion Results
====================================================
           Model:                      OLS
           Method:             Least Squares
           Date:      29-Jun-2021 17:17:08
           Time:                   0.007 s
           R-squared:                0.984
           Adj. R-squared:           0.984
           F-statistic:              46073.89
           Prob (F-statistic):
====================================================
        coef        std err         t         P>|t|
     -0.35065     0.099122      -3.5376   0.00041647
     -0.15432     0.018966      -8.1368    8.605e-16
       1.1334     0.019961       56.781            0


b =

     -0.35065
     -0.15432
       1.1334
```

Users may check the feasibility of the regression by figures in the printed chart. One direct way is to check the `t value` of the regress coefficients. As for the linear OLS above, the first coefficient has the `t value` of -3.5376, meaning that the possibility of P greater than $|t|$ is $2.37 \times 10^-5 << 0.05$. This suggests that the regression is feasible. If the value is greater than 0.05, user may rethink about the feasibility of this regression.

3 Data Hub

CoVEMDA toolbox is designed to utilize the data sourced from COVID-EMDA+. COVID-EMDA+ is specifically designed to track the potential impacts of COVID-19 on the existing U.S. electricity markets. Many different data sources are merged and harmonized here in order to enable further interdisciplinary researches. Historical data dating back to 2017 are included as time-series benchmarks.

3.1 Data Category

Five major kinds of data are included in the data hub. For some categories, multiple data sources are carefully gathered to ensure accuracy through cross-checking or backup.

- Electricity market data. This part includes the generation mix, metered load profiles, day-ahead locational marginal prices data, as well as day-ahead load forecasting, congestion price, forced outage and renewable curtailment data.

  Source: CAISO, MISO, ISO-NE, NYISO, PJM, SPP, ERCOT, EIA, EnergyOnline.

- Public health data. This part includes the COVID-19 confirmed cases, deaths data, infection rate and fatal rate.

  Source: John Hopkins CSSE.

- Meteorological data. This part includes temperature, relative humidity, wind speed and dew point data. Typical weather stations are selected according to their geological locations and data quality.

  Source: Iowa State Univ IEM, NOAA.

- Mobile device location data. This part includes social-distancing data and patterns of visits to Point of Interests (POIs). These data are derived by aggregating and processing the real-time GPS location of cellphone users by Census Block Group.

  Source: Mobility Data From SafeGraph.

- Satellite images. This refers to the Night Time Light (NTL) Satellite data that include the raw satellite images taken at night time in each area. Due to unpredictive cloudy weather, the clear images are limited and only used for visualization.

  Source: NTL Images from NASA.

3.2 Folder Structure

Our data hub mainly contains five folders: source data, released data, supplementary resources, parser codes, and quick start tutorials. The following is a quick navigation:

1. All data source files are archived in folder `data_source`, while the cleaned, processed data are stored in folder `data_release`.

   Latest updates of the released data are properly classified by location. The file name convention is: `<market>_<area>_<category>.csv`, e.g., `nyiso_nyc_load.csv` is a dataset of the load profiles in New York City from 2017 to present.

2. Supplementary resources of several publication works and relevant projects including a series of extended data sources and analysis codes could be found in folder `supplementary`.

3. Basic parser codes (written in Python) to handle the source data are implemented in folder `parser`.

4. Some quick start for the data hub is also accessible in folder `startup` and `supplementary`.

3.3 Field Description

As mentioned before, `data_release` folder contains all the cleaned and processed data. The file name follows the convention of `<market>_<area>_<category>.csv`. All the files are organized in a wide csv table. Here are some tips to further demonstrate the data.

3.3.1 Electricity Market Data

- Field *date*: list all the date from 2017 to present. Format: *YYYYmmdd*
- Field *fuel*: only in generation mix dataset, represent the different fuel source used by generators. Possible values: *coal, gas, oil, nuclear, hydro, wind, solar, export, other.* Different definitions in different electricity markets are harmonized.
- Field *kind*: only in meteorological dataset, represent the type of measurement that is recorded in each corresponding row. Possible value: *dwpc (dew point temperature), relh (relative humidity), sped (wind speed), tmpc (temperature).*
- Field *HH:MM*: represent the hourly time slot. For example, column "08:00" records the data of period 8AM to 9AM.

3.3.2 COVID-19 Data

- Field *date*: list all the date from Jan. 23, 2020 to present. Format: *YYYYmmdd*
- Field *accum_confirm*: accmulated confirmed case number recorded for each date.
- Field *new_confirm*: newly confirmed case number for each date.
- Field *infect_rate*: infection ratio calculated for each date.
- Field *accum_death*: accmulated confirmed death number recorded for each date.
- Field *new_death*: newly confirmed death number for each date.
- Field *fatal_rate*: fatal rate calculated for each date.

3.3.3 Visit Patterns to Point of Interests

- Field *date*: list all the date from Dec.30, 2019 to present. Format: *YYYYmmdd*
- Field *Restaurant_Recreaction*: daily total number of visits to Restaurant and Recreaction places.
- Field *Grocery_Pharmacy*: daily total number of visits to Grocery and Pharmacy places.
- Field *Retail*: daily total number of visits to Retail places.

3.3.4 Social Distancing

- Field *date*: list all the date from Jan. 1, 2020 to present. Format: *YYYYmmdd*
- Field *completely_home_device_count_percentage*: percentage of devices that stay at home 24 hours out of all devices.
- Field *median_home_dwell_time_percentage*: median proportion of home dwell time in one day.
- Field *part_time_work_behavior_devices_percentage*: percentage of devices that go to workplaces for 3 to 6 hours out of all devices.
- Field *full_time_work_behavior_devices_percentage*: percentage of devices that go to workplaces for more than 6 hours out of all devices.

- Field *completely_home_device_count*: count of devices that stay at home 24 hours.
- Field *device_count*: total count of devices.
- Field *part_time_work_behavior_devices_count*: count of devices that go to workplaces for 3 to 6 hours.
- Field *full_time_work_behavior_devices_count*: count of devices that go to workplaces for more than 6 hours.

3.4 Quality Control

To provide reliable and easy-to-use dataset, a series of data checking and cleaning procedures are implemented. Outlier detection and missing data recovery are two key points for the data quality control, and in most cases, outliers or missing data can be easily handled by considering backup data and historical trend, and the whole procedure is shown in Figure 3-1.

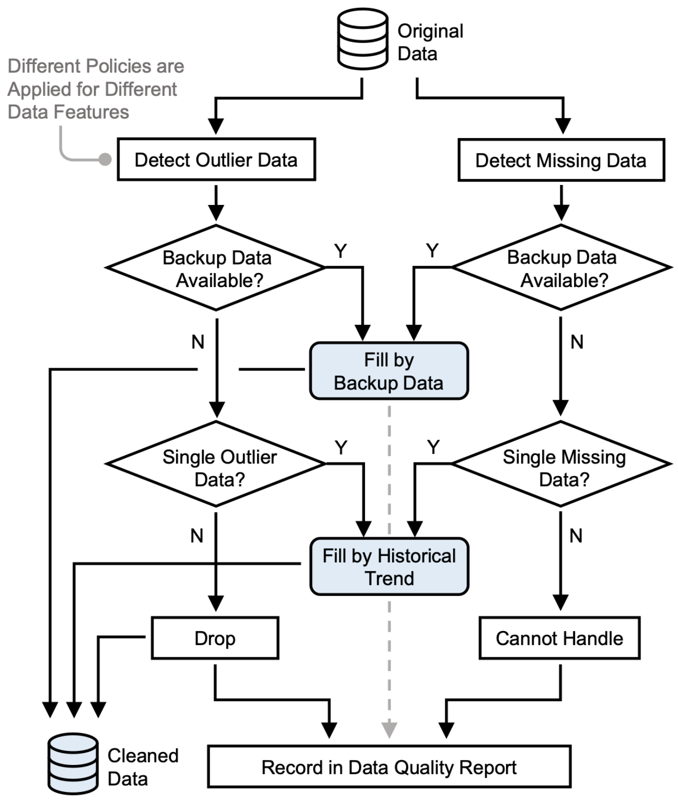

Figure 3-1: Procedure of Data Quality Control.

- Single missing data (most frequent) are filled by linear interpolation. For consecutive missing data (for example, consecutive missing dates, which are very rare), data from the EIA or EnergyOnline are carefully supplemented.
- Outlier data samples are automatically detected when they are beyond 5 times or below 20% of the associated daily average value. Exceptions such as price spikes and negative prices in LMP data are carefully handled.
- Duplicate data are dropped, only the first occurrence of each data sample is kept.

However, there are some cases that cannot be properly handled, e.g., missing data observations for an entire day due to device failures. We thus record these situations in a quality report on Github. Users are recommended to check the report to get full knowledge of the released data before any usage of the data hub. Table 3-1 shows a snapshot of this report on June 6, 2021.

Table 3-1: Data Quality Report (Snapshot on June 6, 2021).

Dataset Name	Market	Area	Category	Missing Dates
caiso_rto_genmix.csv	CAISO	RTO Level	Generation Mix	Sep 23 - Oct 3, 2019
caiso_rto_lmp.csv	CAISO	RTO Level	Day-Ahead LMP	Jan 1 - Jan 18, 2017; Oct 5, Oct 9, 2020
caiso_la_load.csv	Near CAISO	Los Angeles	Hourly Load	Nov 5, 2017; Jan 11 - 12, Jun 29 - 30, Jul 1 - 9, Sep 27, Nov 4, 2018; Mar 10, Nov 3, 2019; Mar 8, Nov 1, 2020; Mar 14, 2021
caiso_la_weather	CAISO	Los Angeles	Weather	Apr 7 - 8, 2021
ercot_rto_load.csv	ERCOT	RTO Level	Hourly Load	Dec 13, 2020
ercot_houston_load.csv	ERCOT	Houston City	Hourly Load	Dec 13, 2020
ercot_houston_weather.csv	ERCOT	Houston City	Weather	Feb 15 - 16, 2020
isone_rto_genmix.csv	ISO-NE	RTO Level	Generation Mix	Apr 30, 2018; Jun 21, Oct 30-31, Dec 06, 2020
isone_rto_load.csv	ISO-NE	RTO Level	Hourly Load	Dec 17, 2020
pjm_rto_genmix.csv	PJM	RTO Level	Generation Mix	Mar 29 - Apr 2, 2017
spp_rto_genmix.csv	SPP	RTO Level	Generation Mix	Mar 29, 2019
spp_kck_weather.csv	SPP	Kansas City	Weather	Sep 22 - 23

4 Baseline Estimation

Baselines are relatively important to quantify the existence and intensity of the pandemic's impacts. This can be properly figured out by comparing the actual observations with the estimated baselines because a reliable baseline is able to provide a control group that haven't experienced COVID-19 [8, 9].

Although estimating baselines is of great importance, there may hardly be a perfect estimation method, and different applications often calls for different methods. For instance, there are many baseline estimators for the power consumption during COVID-19. One classical method is selecting similar dates in the history, and another advanced option is using ensemble backcast model. This model utilizes deep neural networks and multi-dimensional historical data for the estimation task.

In order to support reliable and ready-to-use estimations of baselines, we collect and develop a few popular methods in CoVEMDA with several flexible designs for further extensions. More technical details will be presented in the following subsections.

In CoVEMDA, a function called `cal_baseline` provides a uniform tool for baseline calculation by either date alignment, week index alignment, or backcast estimation. In addition, CoVEMDA offers a function `plot_baseline` for visualizing the baseline results of different estimators. This function calculates the load baseline with different estimation methods by `cal_baseline`, and plots them in one graph. Unlike `cal_baseline`, here multiple methods can be appointed for comparison. See Subsection 4.6.1 and Subsection 4.6.2 for more details.

4.1 Date and Week Alignment

Date alignment for baseline estimation refers to calculating the load baseline by searching for the load data of the exact same date and month in the previous year. This calculation is performed by CoVEMDA function `cal_baseline_by_dates`. See Subsection 4.6.3 for more details.

Week index alignment for baseline estimation refers to calculating the load baseline by searching for the load data of the exact same week and day of week in the previous year. This calculation is performed by CoVEMDA function `cal_baseline_by_week_index`. See Subsection 4.6.4 for more details.

4.2 Ensemble Backcast Estimation

The **Ensemble Backcast Model** is used to estimate the electricity consumption profile in the absence of COVID-19 pandemic, so that the comparison between the model and actual metered electricity consumption can be used to quantify the impact of COVID-19.

A backcast model is expressed as a function that maps potential factors that may affect electricity consumption level, including weather variables (such as temperature, humidity and wind speed), date of year, and economic prosperity (yearly GDP growth rate) to the estimated electricity consumption. Given a group of backcast models, ensemble forecasting is widely recognized as the best approach to providing rich interval information[10]. A group of backcast models for the daily average electricity consumption can be described by

$$\hat{L}_{md} = \frac{1}{N}\sum_{i=1}^{N} \hat{f}_i(C_{md}, T_{mdq}, H_{mdq}, S_{mdq}, E), \forall m, d$$

where C_{md} is the calendar information including month, day, weekday, and holiday flag. $\hat{L}_{md}$ is the estimated daily average electricity consumption for month m and day d. C_{md} is the calendar information including month, day, weekday, and holiday flag. $\hat{L}_{md}$ is the estimated daily average electricity consumption for month m and day d. $\hat{f}_i$ is the i-th backcast model. $T_{mdq}, H_{mdq}, S_{mdq}$ are temperature, humidity and wind speed within the selected quantiles q. E is the estimated GDP growth rate.

We typically select 25%, 50% (average value), 75% and 100%(maximium) quantiles, and the final inputs should be decided based on the the data after extensive testing.

In CoVEMDA, this calculation is realized by the function `cal_baseline_by_backcast`. See Subsection 4.6.5 for more details.

4.3 Abnormal Price Index

Locational marginal prices play a crucial role in electricity markets and contribute to balancing generation and demand efficiently. Since these prices are relatively stochastic in nature, it is unsatisfactory to analyze them in the same way as electricity demand[6]. Thus, the abnormal price index is developed to provide a more reliable baseline for price analysis, which is defined as follows,

$$I(\lambda_{ymdt}) = |2\hat{F}_m(\lambda_{ymdt}) - 1|, \forall y, m, d, t,$$

where $I(\cdot)$ is the proposed index that lies between zero and one, and λ_{ymdt} is the day-ahead locational marginal price of year y, month m, day d, and hour t. $\hat{F}_m(\cdot)$ is the cumulative distribution function for the prices in month m[7]. The proposed index quantifies the abnormality of a typical price, and a larger index value represents a more unusual observation.

The statistical meaning of the above index is explained by the probability density function shown in Figure 4-1. For a given price, the index donates a possibility that measures how close this price is to the mean value. For example, if a price observation is located within the 25% and 75% quantiles, we obtain an index value below 0.5, which is considered normal.

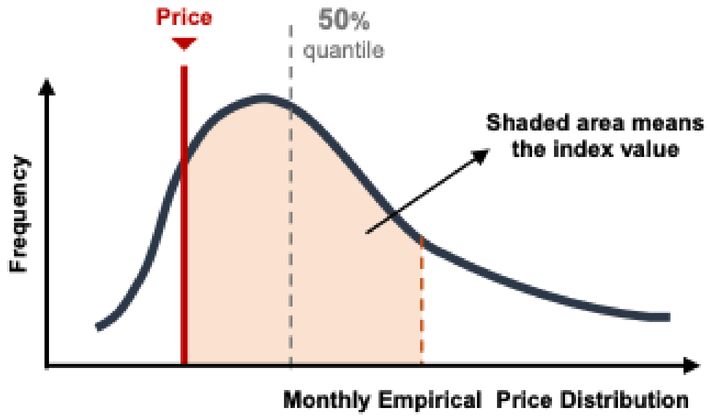

Figure 4-1: Statistical illustration of the proposed abnormal price index. The basic idea is to show the distance between a price observation and the mean value. This index can eliminate some stochastic factors when assessing price changes. The index value lies in [0,1], and a larger value means a higher possibility of abnormality.

In CoVEMDA, the proposed abnormal price index is calculated by function `cal_abnormal_price_index`. More details are available in Subsection 4.6.10.

4.4 Price Distribution Change Across Different Areas

To conduct a cross-market comparison, a Wasserstein probability distance [11][12] is used as metric to quantify the price change during the pandemic. The price change for year y can be formulated as follows:

$$s_y = \text{WD}(\text{Vec}(\lambda_{ymdt}) - \text{Vec}(\lambda_{hist})), \quad \forall y$$

where s_y donates the price change for year y that is calculated by a Wasserstein distance function $\text{WD}(\cdot)$. $\text{Vec}(\cdot)$ is a vectorization function to place all the price data from March to mid-July in a one-dimensional array. λ_{hist} represents all the historical price data accordingly.

In CoVEMDA, the proposed price distribution distance is calculated by function `cal_distribution_diff`. More details are available in Subsection 4.6.11.

[6]the uncertainty intervals will be very wide.

[7]Which are more stable for analyzing these price changes. See [7] for theoretical analysis.

4.5 Excess Change Rate of Renewables

One simple way to quantify the changes of renewable share during the COVID-19 pandemic is to make a linear assumption of its growth rate. Based on the assumption, the **excess change rate** is defined as the relative change rate between the observed renewable share and this estimation:

$$\eta = \left(\frac{r_{2020}}{2r_{2019} - r_{2018}} - 1 \right) \times 100\%$$

where η is the excess change rate, and $r_{2018} \sim r_{2020}$ are the market shares of renewable energy observed in 2018, 2019 and 2020.

In CoVEMDA, the proposed excess change rate is calculated by the function `cal_excess_change_rate`. More details are available in Subsection 4.6.12.

4.6 Functions

4.6.1 `cal_baseline`

Function Definition

```
function t_output=cal_baseline(market,varargin)
```

CoVEMDA function `cal_baseline` provides a uniform tool for baseline calculation by either date alignment, week index alignment or backcast estimation. By default, this function returns the load baseline (as a table or several tables if multiple quantiles are calculated) of specified ISO or city (if `City` is specified) by backcast estimation from 2020-02-01 to 2020-06-30. The user needs to specify the value of name-value pair argument `Method` to alter calculation method (`'date', 'week', 'backcast'`), or `DateRangeCovid` to change the output date range.

In practice, since this function calls the following low-level functions concerning baseline estimation, all the other arguments are passed on to them. For instance, to train the ensemble backcast model with specified model numbers before estimating baseline, the user may call the function like this:

Command Window

```
fx >> cal_baseline('nyiso','TrainModel',1,'ModelNum',[10,3]);
```

Although argument `TrainModel` and `ModelNum` belong to low-level function `cal_baseline_by_backcast`, it is recommended to specify them when calling high-level function `cal_baseline`. See Table 4-1 for a full list of arguments.

Table 4-1: Cal Baseline Options

Name	Type	Description
Method	character array string	Method for baseline estimation, specified as `'date'`, `'week'` or `'backcast'`. *Default* – `'backcast'`.
DateRangeCovid	datetime array string array cell array of character vectors	Range of time for baseline estimation. *Default* – `{'2020-02-01','2020-06-30'}` *Example* – `{'2020-04-01','2020-04-30'}` – `'2020-04-01'`
Others		See Section 4.6.3, Section 4.6.4 and Section 4.6.5.

4.6.2 plot_baseline

Function Definition

```
function gr_handles=plot_baseline(market,varargin)
```

CoVEMDA offers a tool for simple visualization of different baseline estimation methods: `plot_baseline`. This function calculates the load baseline with different estimation methods by `cal_baseline`, plots them in one graph, and returns the handles of graphic objects. The user can appoint the methods to be plotted by specifying the value of name-value pair argument `Method`. Unlike `cal_baseline`, here the parameter value can be multiple method names in the form of cell arrays or string arrays, etc. If so, `cal_baseline` is called for each estimation method.

The user can specify the bin and method for resampling data of the plot by argument `ResampleBin`, `ResampleMethod` as well, just like the visualization functions in Section 6. See Table 4-2 for a full list of arguments.

Table 4-2: Plot Baseline Options

Name	Type	Description
Method	cell array of character vectors string array	Methods for baseline estimation, specified as `'date'`, `'week'` and(or) `'backcast'`. *Default* – `["backcast", "date", "week"]`. *Example* – `'backcast'`
ResampleBin	string character vectors	Binning scheme for resampling. *Default* – `'none'`.
ResampleMethod	string character vectors	Computation method for resampling. *Default* – `'mean'`.
Others		See Section 4.6.1.

4.6.3 cal_baseline_by_dates

Function Definition

```
function t_output=cal_baseline_by_dates(market,dr_Covid,varargin)
```

If `Method` is specified as `date`, function `cal_baseline` calculates the load baseline by searching for the load data of the exact same date and month in the previous year, for every day in `DateRangeCovid`. This calculation process is performed by function `cal_baseline_by_dates`. It returns a table containing the dates and load baseline data.

The only additional argument here is `City`, specified as one of the seven abbreviations: `'la'`, `'houston'`, `'boston'`, `'nyc'`, `'chicago'`, `'phila'` and `'kck'`, or `'rto'` (default) if the user needs the result for the entire market.

4.6.4 cal_baseline_by_week_index

Function Definition

```
function t_output=cal_baseline_by_week_index(market,dr_Covid,varargin)
```

If `Method` is specified as `week`, function `cal_baseline` calculates the load baseline by searching for the load data of the exact same week and day of week in the previous year, for every day in `DateRangeCovid`. This calculation process is performed by function `cal_baseline_by_week_index`. It returns a table containing the dates and load baseline data.

The only additional argument here is `City` as well, specified as one of the seven abbreviations: `'la'`, `'houston'`, `'boston'`, `'nyc'`, `'chicago'`, `'phila'` and `'kck'`, or `'rto'` (default) if the user needs the result for the entire market.

4.6.5 `cal_baseline_by_backcast`

Function Definition

```
function t_YPreds=cal_baseline_by_backcast(market,dr_Covid,varargin)
```

If `Method` is specified as `backcast`, function `cal_baseline` calculates the load baseline by either training new models or using existing models with function `cal_baseline_by_backcast`. If only one single quantile is appointed, this function returns a table containing the dates and load baseline data like the others. However, if multiple quantiles are appointed, this function returns a table matching the quantiles with tables containing dates and baseline correspondingly. All parameters are listed in Table 4-3.

In general, this function consists of 3 or 6 steps (depends on whether to train models or not). If the user chooses to train a new model, the function

1. imports training data and splits training and testing set (`backcast_import_data`);
2. trains two-layer deep neural networks using training set with validation (`backcast_scan_training`);
3. tests all the models and selects the best ones using testing set (`backcast_test_model`);
4. saves trained models and summary (`basic_backcast_save_file`);
5. imports data for baseline estimation (`backcast_import_data`);
6. calculates the load baseline with quantiles (`backcast_scan_prediction`).

If the user chooses not to train models but to use pre-trained models, the function

1. imports existing models (`basic_backcast_import_models`);
2. imports data for baseline estimation (`backcast_import_data`);
3. calculates the load baseline with quantiles (`backcast_scan_prediction`).

4.6.6 `backcast_import_data`

Function Definition

```
function [t_XTrainandtest,t_YTrainandtest,t_XTrainandvalid,t_YTrainandvalid
    ,t_XTest,t_YTest]=backcast_import_data(market,dr_Trainandtest,varargin)
```

Function `backcast_import_data` is the subfunction of `cal_baseline_by_backcast`, in charge of importing data for training, validation and testing of the ensemble backcast model. `t_XTrainandtest` and `t_YTrainandtest` are the training (including validation) and testing set, and the others are training-validation set and testing set separated from each other, i.e., the union of the last four equals the first two.

The only additional argument here is `City`, specified as one of the seven abbreviations: `'la'`, `'houston'`, `'boston'`, `'nyc'`, `'chicago'`, `'phila'` and `'kck'`, or `'rto'` (default) if the user needs the result for the entire market.

Table 4-3: Cal Baseline By Backcast Options

Name	Type	Description
City	string character array	Abbreviation of the target city or `'rto'` for the entire market. *Default* – `'rto'`.
TrainModel	integer	Train (1) or not train (0) models. *Default* – 0.
ImportNum	integer or `'latest'`	The id of the existing model to be imported. Only effective when TrainModel=0. *Default* – `'latest'`.
DateRangeTrain	datetime array string array cell array of character vectors	Range of time for model training. Only effective when TrainModel=1. *Default* – `{'2018-02-01','2020-01-31'}`. *Example* – `{'2018-01-01','2019-12-31'}` – `["2018-01-01","2019-12-31"]`
ModelNum	two-element numeric vector	The number of models to be tested and selected. Only effective when TrainModel=0. *Default* – `[800, 200]`.
SaveModel	integer	Indicator to save trained models in MATLAB .asv files. Save(1) or not save(0). Only effective when TrainModel=0. *Default* – 1.
SaveModelONNX	integer	Indicator to save trained models in .onnx files. Save(1) or not save(0). Only effective when TrainModel=0. *Default* – 0.
SaveSummary	integer	Indicator to save summary of trained models. Save(1) or not save(0). Only effective when TrainModel=0. *Default* – 1.
Others		See Section 4.6.6, Section 4.6.7, Section 4.6.8 and Section 4.6.9.

4.6.7 `backcast_scan_training`

Function Definition

```
function t_net=backcast_scan_training(t_XTrainandvalid,t_YTrainandvalid,
    model_num,varargin)
```

Function `backcast_scan_training` is the subfunction of `cal_baseline_by_backcast`, in charge of training the deep neural networks. This function trains `model_num` deep neural networks with two hidden layers and random number of hidden units, determined by argument `numHiddenUnits` and `RandVariation`. The return value of this function is a table of all the trained models, along with their parameters, validation accuracy and corresponding scalers. All parameters are listed in Table 4-4.

4.6.8 `backcast_test_model`

Function Definition

```
function [t_net_all,t_net_sel]=backcast_test_model(t_net_all,t_XTest,
    t_YTest,model_sel,varargin);
```

Function `backcast_test_model` is the subfunction of `cal_baseline_by_backcast`, in charge of selecting the best deep neural networks. This function selects the best `model_sel` models using the testing set.

Table 4-4: Backcast Scan Training Options

Name	Type	Description
numHiddenUnits	two-element numeric vector	The number of hidden units in the two full connection layers. Only effective when TrainModel=0. *Default* – `[32, 64]`.
TrainVerbose	integer	Indicator to display training progress information. No display(0), brief display(1) or detailed display(2). Only effective when TrainModel=0. *Default* – 1.
RandVariation	double value in (0,1)	Relative range (1-RandVariation, 1+RandVariation) of random variation of model parameters numHiddenUnits for scan training. Only effective when TrainModel=0. *Default* – 0.2.

`t_net_sel` is a table of the selected models, and `t_net_all` is still the table of all models, combined with the information (testing accuracy, sign of selection) of the test.

The only additional argument here is `TrainVerbose`, which is the same as the one in Section 4.6.7.

4.6.9 `backcast_scan_prediction`

Function Definition

```
function t_YPreds=backcast_scan_prediction(t_net,t_XPred,varargin)
```

Function `backcast_scan_prediction` is the subfunction of `cal_baseline_by_backcast`, in charge of calculating baseline using the ensemble backcast model. It returns a table containing the dates and load baseline data. All parameters are listed in Table 4-5.

Table 4-5: Backcast Scan Prediction Options

Name	Type	Description
Quantile	numeric vector in (0,1)	Quantiles to be calculated in scan prediction. *Default* – `[0.1, 0.25, 0.5, 0.75, 0.9]`. *Example* – 0.5
PredictVerbose	integer	Indicator to display prediction progress information. No display(0) or brief display(1). *Default* – 1.

4.6.10 `cal_abnormal_price_index`

Function Definition

```
function output=cal_abnormal_price_index(market,varargin)
```

Function `cal_abnormal_price_index` calculates the proposed abnormal index mentioned in Subsection 4.3. The output is a table containing abnormal price index and corresponding dates. All parameters are listed in Table 4-6.

4.6.11 `cal_distribution_diff`

Table 4-6: Cal Abnormal Price Index and Cal Distribution Diff Options

Name	Type	Description
DateRange	datetime array string array cell array of character vectors	Range of time for index calculation. *Default* – `{'2020-03-01','2020-07-15'}`. *Example* – `{'2020-01-01','2020-12-31'}` – `["2020-01-01","2020-12-31"]`
City	string character array	Abbreviation of the target city or `'rto'` for the entire market. *Default* – `'rto'`.

Function Definition

```
function output=cal_distribution_diff(market,varargin)
```

Function `cal_distribution_diff` provides an indicator for the difference of price distribution with data from the previous year as baseline. It adopts Wasserstein distance as the measurement method. All parameters are listed in Table 4-6.

4.6.12 `cal_excess_change_rate`

Function Definition

```
function output=cal_excess_change_rate(market,varargin)
```

Function `cal_excess_change_rate` calculates the proposed abnormal index in Section 4.5. The return value is the proposed excess change rate of the specified market. All parameters are listed in Table 4-7.

Table 4-7: Cal Excess Change Rate Options

Name	Type	Description
DateRange	datetime array string array cell array of character vectors	Range of time for index calculation. *Default* – `{'2017-01-01','2020-07-15'}`. *Example* – `{'2017-01-01','2020-12-31'}` – `["2017-01-01","2020-12-31"]`
City	string character array	Abbreviation of the target city or `'rto'` for the entire market. *Default* – `'rto'`.

4.7 Examples

In this section, we introduce the usage of the functions above with several examples. It is recommended to use feature-oriented high-level functions (see A.2), and the following explanations and instructions are designed for them.

All high-level functions mentioned above (`cal_baseline`, `plot_baseline`, `cal_abnormal_price_index`, `cal_distribution_diff` and `cal_excess_change_rate`) have similar input forms and can be used similarly. To be specific, the first input argument (market name) is compulsory, and all the other input arguments are optional. The market name is specified as one of the seven terms:

`'caiso' | 'ercot' | 'isone' | 'miso' | 'nyiso' | 'pjm' | 'spp'`

As long as the market name is given, a basic version of the desired result can be simply obtained, which is illustrated in Section 2.4.1. All the other input arguments should be in the form of **name-value pairs** for the function to correctly recognize.

```
Command Window
cal_baseline(market,'argument_name1',argument_value1,'argument_name2',
    argument_value2,...);
```

For example, if the user needs to train a new ensemble backcast model with less networks for baseline estimation:

```
Command Window
>> t_retrain=cal_baseline('caiso','TrainModel',1,'ModelNum',[10,8])
Training, 10 models in total.
--Training Model #1, number of hidden units: 34  60.
--Model #1 trained, MAPE: 2.69%
--Training Model #2, number of hidden units: 29  52.
--Model #2 trained, MAPE: 3.04%
...
--Training Model #9, number of hidden units: 33  54.
--Model #9 trained, MAPE: 3.27%
--Training Model #10, number of hidden units: 35  59.
--Model #10 trained, MAPE: 2.86%
Testing, 10 models in total.
--Running Model #1.
--Running Model #2.
...
--Running Model #9.
--Running Model #10.
Predicting, 8 models in total.
--Running Model #9.
--Running Model #7.
--Running Model #5.
--Running Model #4.
--Running Model #3.
--Running Model #2.
--Running Model #1.
--Running Model #10.

t_retrain =

  5x2 table

    quantile        result
    --------    --------------

      0.1       {151x25 table}
     0.25       {151x25 table}
      0.5       {151x25 table}
     0.75       {151x25 table}
      0.9       {151x25 table}
```

As illustrated in Section 4.6.5, this function returns a table whose first column is the quantiles (0.1, 0.25, 0.5, 0.75, 0.9), and each element in the second column is a table containing dates and baseline for corresponding quantile. Summary of the newly trained model can be found at:

`<root_directory>\data_temp\backcast\models\caiso_rto\Model_<timestamp>\Summary.csv`

For function `plot_baseline`, its legal parameters include both `cal_baseline` options and modifications of the plot. Like the functions in Section 6, the user can change property values of the chart after calling this function. For example, if the user needs to visualize different baselines: (command output omitted)

```
Command Window
>> handles=plot_baseline('nyiso','DateRangeCovid','2020-04-01');
fx >> handles{6}.Color='#d95319';
```

Add the observed demand data to the graph and adjust x-axis labels:

```
Command Window
>> plot_demand('nyiso','DateRange','2020-04-01','DisplayName',
   'observed','Color','#edb120');
>> ax=gca;
fx >> ax.XTickLabel=strcat(string([0:2:22,23]),":00");
```

The result is displayed in Figure 4-2. For the baseline estimated by ensemble backcast model, the intervals are represented with areas of different brightness, indicating the distribution, which is the same as `plot_quantile` in Section 6.5.8.

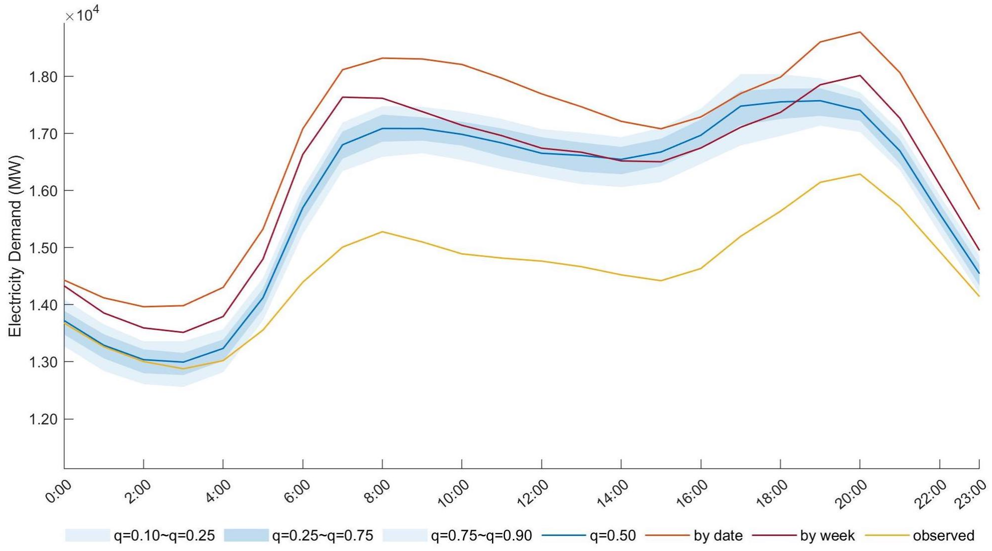

Figure 4-2: Example: Baseline and Observed Demand of NYISO

5 Regression Analysis

Regression is powerful to answer a variety of questions regarding the correlation and causality. It is possible, for example, to explore the underlying relationship between demand drop during COVID-19 and other cross-domain factors of public health, social distancing, and retail mobility. In this section, we hereby introduce two popular regression methods to study the correlated and causal relationships during COVID-19.

It should be noted that the toolbox supports a flexible syntax for regression, and users may choose from a built-in collection of linear, interaction, quadratic, or pure-quadratic methods. But external data inputs and user-defined functions are allowed for extensions.

5.1 Ordinary Least Squares Regression

Ordinary least squares (OLS) models[13] includes linear and non-linear models and offers multiple choices for model fitting. Besides, CoVEMDA also support correlation analysis for variables either from COVID-EMDA+ or from external sources.

OLS model is calibrating by minimizing the regression residuals, shown as follows:

$$minf(x) = \sum_{i=1}^{m} L_i^2(x) = \sum_{i=1}^{m} [y_i - f(x_i, \omega_i)]^2, \tag{5.1}$$

where $L_i(x)$ is the residual function. If $L_i(x)$ is a linear function about x, the optimization problem is called 'linear least squares problem', while nonlinear ones are addressed as 'nonlinear least squares problems'.

5.2 Vector Autoregression

Vector autoregression (VAR) is a stochastic process model that can be used to capture the linear correlation between multiple time-series data [14]. This model can be extended to be restricted vector autoregression when some parameters are set to be zeros. Both models are powerful tools for multivariate time series analysis and have been widely adopted in economics [15] and electricity markets [16].

A typical expression of a p-order VAR model is shown as follows:

$$X_t = C + A_1 X_{t-1} + \cdots + A_p X_{t-p} + E_t, \tag{5.2}$$

where

$$A_i = \begin{bmatrix} a_{1,1}^i & a_{1,2}^i & \cdots & a_{1,n}^i \\ a_{2,1}^i & a_{2,2}^i & \cdots & a_{2,n}^i \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}^i & a_{n,2}^i & \cdots & a_{n,n}^i \end{bmatrix}, X_t = \begin{bmatrix} x_t^1 \\ x_t^2 \\ \vdots \\ x_t^n \end{bmatrix}, C = \begin{bmatrix} c^1 \\ c^2 \\ \vdots \\ c^n \end{bmatrix}, E_t = \begin{bmatrix} e_t^1 \\ e_t^2 \\ \vdots \\ e_t^n \end{bmatrix}, \tag{5.3}$$

in which A_i is the regression matrix, x_t^1 represents the target output variable at time t, namely the reduction in electricity consumption we wish to model, $x_t^2, \cdots, x_t^n$ represent the selected $n-1$ parameter variables including confirmed case numbers, stay-at-home population, median home dwell time rate, population of on-site workers, mobility in the retail sector and etc., C and E_t are respectively column vectors of intercept and random errors, and the time notation $t-p$ represents the p-th lag of the variables.

The full procedure of building the restricted VAR model mainly contains four steps as follows, including pre-estimation preparation, restricted VAR model estimation, restricted VAR model verification, and post-estimation analysis.

5.2.1 Pre-estimation Preparation

- Data Preprocessing: Several datasets are collected to calculate the inputs of restricted VAR model, including electricity market data, meteorological data, number of COVID-19 cases, and mobile device location data. We take logarithms of several variables, including electricity consumption reduction, new daily confirmed cases, stay-at-home population, population of full-time on-site workers, population of part-time on-site workers, and mobility in the retail sector, while only keeping the original value of the median home dwell time rate.

- Augmented Dickey-Fuller (ADF) Test[17]: Test whether a time-series variable is non-stationary and possesses a unit root.
- Cointegration Test: Test the long-term correlation between multiple non-stationary time-series.
- Granger Causality Wald Test: Estimate the causality relationship among two variables represented as time-series.

5.2.2 VAR Model Estimation

- Ordinary Least Square (OLS): We impose constraints on the OLS to eliminate any undesirable causal relationships between variables.

5.2.3 VAR Model Verification

- ADF Test: Verify if the residual time-series are non-stationary and possess a unit root.
- Ljung-Box Test: Verify the endogeneity of the residual data that may render the regression result untrustworthy.
- Durbin-Watson Test: Detect the presence of autocorrelation at log 1 in the residuals of the Restricted VAR model.
- Robustness Test: Test the robustness of the Restricted VAR model against parameter perturbations.

5.2.4 Post-estimation Analysis

- Impulse Response Analysis: Describe the evolution of the Restricted VAR model's variable in response to a shock in one or more variables.
- Forecast Error Variance Decomposition: Aid in the interpretation of the Restricted VAR model by determining the proportion of each variable's forecast variance that is contributed by shocks to the other variables.

5.3 Functions

5.3.1 `OLS`

Function Definition

```
function output=OLS(table,varnum,varargin)
```

Function `OLS` gives a solution to the regression model in Section 5.1. The output is a printed table containing regression vector and correlation analysis. All parameters are listed in Table 5-1.

5.3.2 `VAR`

Function Definition

```
function VAR(table,varnum,order)
```

Function `VAR` gives a solution to the regression model in Section 5.2. The output is a printed table containing regression vector and correlation analysis. Unlike OLS regression model, VAR doesn't distinguish dependent variables and independent variables. It gives an evaluation table of the parameters, listing the coefficient values between each two variables. All parameters are listed in Table 5-2.

Table 5-1: OLS Parameters and Options

Name	Type	Description
table	integer	Table of variable vectors for regression
varnum	vector	Numbers of input variable vectors
output	vector	Regression results
regression model	string character array	Term options for model training. Including constant, linear, interaction and squared terms. *Default* – `'linear'`. *Example* – `'interaction'` – `'quadratic'` – `'purequadratic'`

Table 5-2: VAR Parameters and Options

Name	Type	Description
table	table	Table of variable vectors for regression
varnum	integer	Numbers of input variable vectors
order	integer	order of the VAR model

5.4 Examples

In this section, we introduce the usage of the functions above with several examples. It is recommended to use feature-oriented high-level functions (see A.2), and the following explanations and instructions are designed for them.

All high-level functions mentioned above (`OLS` and `VAR`) have similar input forms and can be used similarly. To be specific, the first input argument (table name) is compulsory, and the basic operation function `basic_read` can help import csv files in format of tables.

5.4.1 OLS example

As for function `OLS`, choosing a regression model is an essential step. Four OLS models are provided as follows, with each formula introduced in Section 5.3.1.

`'linear' | 'interaction' | 'quadratic' | 'purequadratic'`

As introduced above in Section 5.3.1, the feature-oriented high-level function `OLS` takes form as follows:

Command Window

```
fx OLS(table,varnum,varargin);
```

Meanwhile, user may probably need basic-operation functions `basic_read` and `basic_exchange_columns`, which take forms as follows:

Command Window

```
fx t = basic_read(filename);
```

Command Window

```
fx output = basic_exchange_columns(table,column1,column2)
```

For example, if user needs to do a regression with data from COVID-EMDA+ data hub, he may do some data-preprocessing before regression. Take file `'caiso_la_patterns.csv'` as an example. Code is provided as follows. Firstly, user may employ the basic-operation function `basic_read` to import data in table format.

Command Window

```
>> t=basic_read('caiso_la_patterns')

t =

  329x4 table

       date       Restaurant_Recreaction    Grocery_Pharmacy      Retail
    __________    ______________________    ________________    __________

    2019-12-30          3.2519e+05             1.0855e+05       6.1559e+05
    2019-12-31          3.1989e+05             1.0976e+05       5.8792e+05
    2020-01-01          2.3363e+05                  51943       3.6671e+05
    2020-01-02          2.9873e+05                  90239        5.481e+05
fx                 ...                                    ...
```

Secondly, `OLS` takes the fist column from table as dependent variables by default and the following columns as independent ones. Accordingly, function `basic_exchange_columns` should be applied to exchange columns of table.

Command Window

```
>> t=basic_exchange_columns(t,1,4)

t =

  329x4 table

      Retail      Restaurant_Recreaction    Grocery_Pharmacy       date
    __________    ______________________    ________________    __________

    6.1559e+05          3.2519e+05             1.0855e+05       2019-12-30
    5.8792e+05          3.1989e+05             1.0976e+05       2019-12-31
    3.6671e+05          2.3363e+05                  51943       2020-01-01
     5.481e+05          2.9873e+05                  90239       2020-01-02
fx               ...                                    ...
```

Last but not least, user may do the regression part.

Command Window

```
>> b=OLS(t,3,'linear')
               OLS Regerssion Results
====================================================
           Model:                      OLS
           Method:             Least Squares
           Date:       29-Jun-2021 11:59:42
           Time:                     0.392 s
           R-squared:                  0.996
           Adj. R-squared:             0.996
           F-statistic:                38471.97
           Prob (F-statistic):
====================================================
        coef          std err          t          P>|t|
      -8028.9         1872.5       -4.2879    2.3789e-05
        1.207       0.018001        67.051   8.9386e-193
       2.0651       0.075663        27.294     3.344e-86


b =

      -8028.9
        1.207
       2.0651
```

5.4.2 VAR example

As for function `VAR`, the regression mode is already set. And as introduced above in Section 5.3.2, the feature-oriented high-level function `VAR` takes form as follows:

Function Definition

```
function VAR(table,varnum,order);
```

Unlike function `OLS`, there is no output for this function. Meanwhile, user may probably need basic-operation functions `basic_read` and `basic_exchange_columns`, which take forms as follows:

Function Definition

```
function t = basic_read(filename);
```

Function Definition

```
function output = basic_exchange_columns(table,column1,column2)
```

For example, if user needs to do a VAR with data of CAISO, which is stored in `caiso_rto_lmp.csv`, some data pre-processing may be necessary before regression. Firstly, user may employ the basic-operation function `basic_read` to import data in table format.

Command Window

```
>> t=basic_read('caiso_rto_lmp')

t =

  1462x25 table

        date        00:00      01:00      02:00                ...
    ----------     ------     ------     ------                ...

    2017-01-19      21.53      21.69      20.86                ...
    2017-01-20      26.46      24.67      23.68                ...
    2017-01-21      30.83      28.37      28.19                ...
    2017-01-22      27.33      26.5       25.14                ...
                 ...                    ...                    ...
```

Secondly, `VAR` takes the fist few columns from table as variables by default, and the other columns remain untouched. Accordingly, function `basic_exchange_columns` should be applied to exchange columns of table.

Command Window

```
>> t=basic_exchange_columns(t,1,4)

t =

  1462x25 table

    02:00      00:00      01:00         date
    ------     ------     ------     ----------                ...

    20.86      21.53      21.69      2017-01-19                ...
    23.68      26.46      24.67      2017-01-20                ...
    28.19      30.83      28.37      2017-01-21                ...
    25.14      27.33      26.5       2017-01-22                ...
             ...                    ...                        ...
```

Last but not least, user may do the VAR part.

Command Window

```
>> VAR(t,3,2)
               OLS Regerssion Results
====================================================
          Model:                      VAR
          Method:            Least Squares
          Date:      29-Jun-2021 16:06:28
====================================================
        coef         std err          t        P>|t|
       2.2366         0.3111       7.1893   1.0376e-12
      0.85631       0.087143       9.8265   4.1469e-22
    -0.073503       0.064774      -1.1348      0.25666
      0.16578        0.11112       1.4919      0.13594
    -0.024319       0.087911     -0.27663       0.7821
    -0.052469       0.064236     -0.81681      0.41417
     0.051594         0.1113      0.46356      0.64304
%% each sub table have 3x2+1=7 rows

        coef         std err          t        P>|t|
       2.8308         0.3498       8.0927   1.2202e-15
       0.3993       0.097985       4.0751    4.847e-05
      0.31999       0.072833       4.3934   1.1966e-05
       0.2698        0.12494       2.1594     0.030985
      -0.1529       0.098849      -1.5468      0.12213
      0.13715       0.072228       1.8989     0.057782
    -0.025913        0.12515     -0.20706      0.83599


        coef         std err          t        P>|t|
       2.6695        0.32137       8.3065   2.2288e-16
      0.57071       0.090021       6.3398   3.0629e-10
     0.031848       0.066913      0.47596      0.63417
      0.36905        0.11479        3.215    0.0013332
    -0.085903       0.090815     -0.94591      0.34435
    -0.022497       0.066357     -0.33903      0.73463
     0.064468        0.11498      0.56071      0.57508
```

6 Scientific Visualization

Visualizing the raw data or statistical results is an intuitive way to convey messages. This also contributes to further explaining of the baselines or regression results that are obtained from above sections. CoVEMDA collects a variety of classical visualization applications, and design specific methodologies accordingly. We carefully balance the requirements of ease-of-use and flexibility by providing various functions of either low- or high- levels.

6.1 Generation Mix

After the COVID-19 outbreak, the total generation as well as generation share of each market in U.S. varied. In order to analyse this change, we may observe the generation data in scale of time or generation fuels. This may offer a new approach for researchers to find out the structural changes of U.S. power grid brought by COVID-19.

In CoVEMDA, two typical tools regarding different scales are provided: `plot_generation_mix` and `plot_renewable_share`. Function `plot_generation_mix` plots the generation mix data in the form of filled areas, and function `plot_renewable_share` plots the renewable share data in the form of a line chart. Both of them obtain relevant data from file `<market_name>_rto_genmix.csv` in the data hub. See Section 6.5.1 and Subsection 6.5.2 for more details.

```
Command Window
fx >> plot_generation_mix('spp');
```

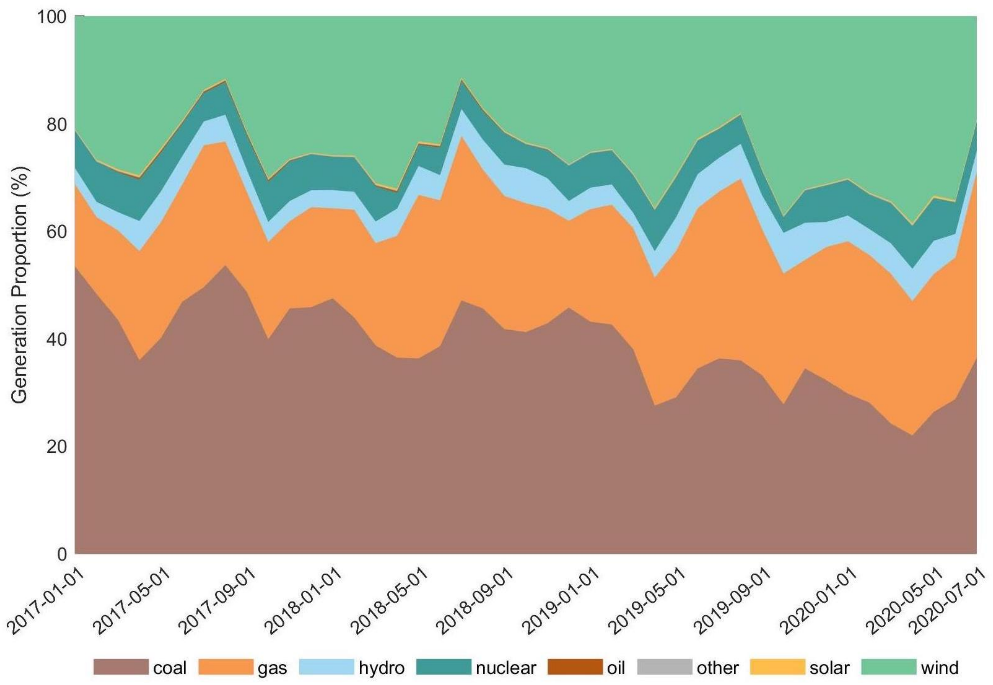

Figure 6-1: Generation Mix of SPP

One possible output of function `plot_generation_mix` with default settings is displayed in Figure 6-1. Different color blocks demonstrate different fuel kinds as shown in legends. The area of each block shows the electric generation of each kind of fuel. In this figure, we may see that changes in generation structure are mainly caused by the increasing quantity of wind power generation.

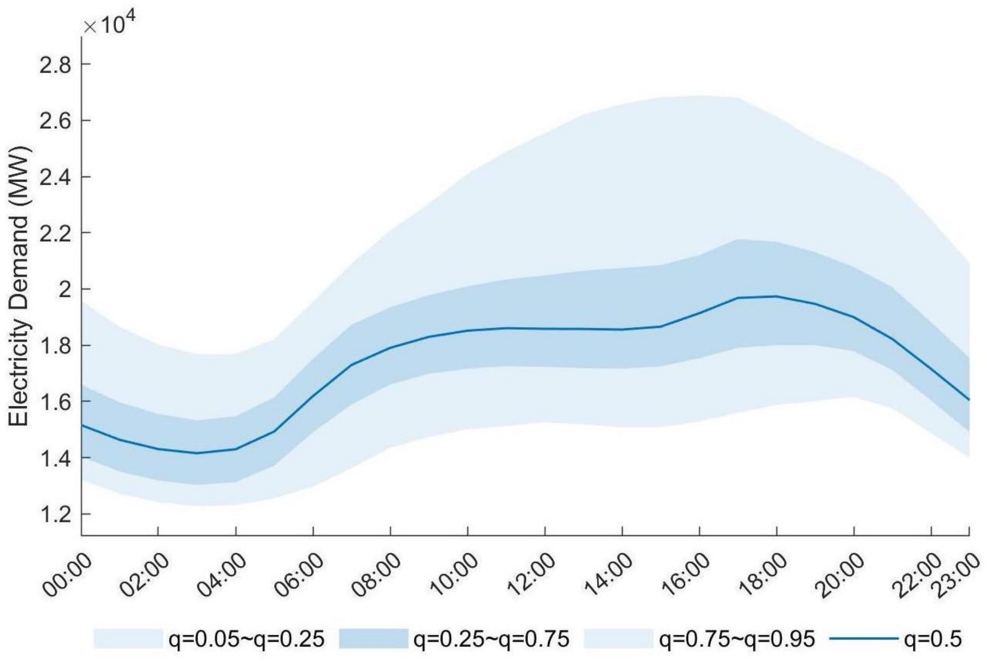

Figure 6-2: Daily Demand Curve of NYISO

6.2 Demand Profiles

In power system analysis, demand profile is a basic but significant data whose changes may induce systematic impacts. One important finding from existing publications is that the demand profiles around the globe may experience a series of severe impacts. Visualization helps to show such changes in a clear way.

In CoVEMDA, two typical tools regarding different time scales are provided for demand analysis: `plot_demand` and `plot_daily_demand_curve`. Function `plot_demand` plots the load data in the form of a line chart, and function `plot_daily_demand_curve` plots the demand distribution curve in the form of multiple areas. Both of them obtain relevant data from file `<market_name>_<city_name>_load.csv` in the data hub. See Subsection 6.5.3 and Subsection 6.5.4 for more details.

Command Window

```
fx >> plot_daily_demand_curve('nyiso');
```

One possible output of function `plot_daily_demand_curve` with default settings is displayed in Figure 6-2. In this plot, five edges of color blocks represent the five quantiles, `[0.05, 0.25, 0.5, 0.75, 0.95]` successively. As there are odd quantiles, the `0.5` quantile is displayed as the central line. The outer light blue area represents the variation of `0.05~0.95`, and the inner less-light blue area represents `0.25~0.75`, to better illustrate the variation of electricity demand in the date range.

6.3 Price Distribution

In this section, we provide two typical tools to display the electricity price in the U.S. After the COVID-19 outbreak, electricity price becomes hard to trace. In order to trace the change of electricity price, we may observe the price data of different markets in the U.S. and organize them into plots. This may offer a new approach for researchers to find out the structural changes of U.S. power grid brought by COVID-19.

In CoVEMDA, two typical tools regarding different scales are provided for price analysis: `plot_price` and `plot_price_distribution`. Function `plot_price` plots the price data in the form of a line chart, and function `plot_price_distribution` plots the price distribution in the form of a histogram. Both of them obtain relevant data from file `<market_name>_<city_name>_lmp.csv` in the data hub. See Subsection 6.5.5 and Subsection 6.5.6 for more details.

6.4 Mobility

Mobility is a factor directly related with the pandemic. Here "mobility" means the number of visits to business premises, indicating the level of commercial activities. During COVID-19, the mobility in various sectors have experienced dramatic reduction, which is one of the first signals of the outbreak.

In CoVEMDA, the mobility for three sectors are provided: `Restaurant_Recreation`, `Grocery_Pharmacy` and `Retail`. Unlike other functions in this section, here the data are exclusively focused on seven major cities: Los Angeles(`la`), Houston(`houston`), Boston(`boston`), New York City(`nyc`), Chicago(`chicago`), Philadelphia(`phila`) and Kansas City(`kck`).

All the mobility data above are obtained from file `<market_name>_<city_name>_patterns.csv` and can be visualized by function `plot_mobility`. See Subsection 6.5.7 for more details.

6.5 Functions

6.5.1 `plot_generation_mix`

Function Definition

```
function ar=plot_generation_mix(market,varargin)
```

Function `plot_generation_mix` plots the generation mix of a market within specified date range. Like `area`, it returns the area objects of the curve for further modification. This function is based on MATLAB function `area`.

By default, this function plots the generation mix of the specified market from 2017-01-01 to 2020-07-15 with month resampling. Additional name-value pair arguments are available for modifications. See Table 6-1 for a full list of arguments.

Table 6-1: Plot Generation Mix Options

Name	Type	Description
DateRange	datetime array string array cell array of character vectors	Range of time. *Default* – `{'2017-01-01','2020-07-15'}`. *Example* – `{'2017-01-01','2020-07-15'}` – `["2020-02-01","2020-04-30"]`
City	string character array	Abbreviation of the target city or `'rto'` for the entire market. *Default* – `'rto'`.
ResampleBin	string character vectors	Binning scheme for resampling. *Default* – `'month'`.
Others		See Section 6.5.8.

6.5.2 `plot_renewable_share`

Function Definition

```
function p=plot_renewable_share(market,varargin)
```

Function `plot_renewable_share` plots the renewable share curve of a market within specified date range. Like `plot`, it returns the line objects of the curve for further modification. This function is based on MATLAB function `plot`.

By default, this function plots the renewable share of the specified market from 2017-01-01 to 2020-07-15, with no resampling. Additional name-value pair arguments are available for modifications. See Table 6-2 for a full list of arguments.

6.5.3 `plot_demand`

Function Definition

```
function p=plot_demand(market,varargin)
```

Function `plot_demand` plots the demand curve of a market within specified date range. Like `plot`, it returns the line object of the curve for further modification. This function is based on MATLAB function `plot`.

By default, this function plots the demand curve of the specified market from 2017-01-01 to 2020-07-15, with no resampling. Additional name-value pair arguments are available for modifications. See Table 6-2 for a full list of arguments.

Table 6-2: Plot Renewable Share, Plot Demand and Plot Price Options

Name	Type	Description
DateRange	datetime array string array cell array of character vectors	Range of time. *Default* – `{'2017-01-01','2020-07-15'}`. *Example* – `{'2017-01-01','2020-07-15'}` – `["2020-02-01","2020-04-30"]`
City	string character array	Abbreviation of the target city or `'rto'` for the entire market. *Default* – `'rto'`.
ResampleBin	string character vectors	Binning scheme for resampling. *Default* – `'none'`.
ResampleMethod	string character vectors	Computation method for resampling. † *Default* – `'mean'`.
Others		See Section 6.5.9.

6.5.4 `plot_daily_demand_curve`

Function Definition

```
function gr_handles=plot_daily_demand_curve(market,varargin)
```

Function `plot_daily_demand_curve` plots the daily demand curve of a market with specified quantiles. It returns all the objects in the plot for further modification. If there are odd quantiles, the central one is displayed as a line object. Otherwise all of them are displayed as area objects. This function is based on MATLAB function `area` and `plot`.

By default, this function plots the daily demand curve of the specified market from 2017-01-01 to 2020-07-15 with the five quantiles representing its variation: `[0.05, 0.25, 0.5, 0.75, 0.95]`. Additional name-value pair arguments are available for modifications. See Table 6-3 for a full list of arguments.

6.5.5 `plot_price`

Function Definition

```
function p=plot_price(market,varargin)
```

Table 6-3: Plot Daily Demand Curve Options

Name	Type	Description
DateRange	datetime array string array cell array of character vectors	Range of time. *Default* – `{'2017-01-01','2020-07-15'}`. Example – `{'2017-01-01','2020-07-15'}` – `["2020-02-01","2020-04-30"]`
City	string character array	Abbreviation of the target city or `'rto'` for the entire market. *Default* – `'rto'`.
Quantile	numeric vector	Cumulative probabilities. *Default* – `[0.05, 0.25, 0.5, 0.75, 0.95]`.
Others		See Section 6.5.10.

Function `plot_price` plots the price curve of a market within specified date range. Like `plot`, it returns the line objects of the curve for further modification. This function is based on MATLAB function `plot`.

By default, this function plots the price of the specified market from 2017-01-01 to 2020-07-15, with no resampling. Additional name-value pair arguments are available for modifications. See Table 6-2 for a full list of arguments.

6.5.6 `plot_price_distribution`

Function Definition

```
function H=plot_price_distribution(market,varargin)
```

Function `plot_price_distribution` plots the price distribution of a market. This function is based on MATLAB function `histogram`.

By default, this function plots the price distribution of the specified market from 2017-01-01 to 2020-07-15. Additional name-value pair arguments are available for modifications. See Table 6-4 for a full list of arguments.

Table 6-4: Plot Price Distribution Options

Name	Type	Description
DateRange	datetime array string array cell array of character vectors	Range of time. *Default* – `{'2017-01-01','2020-07-15'}`. *Example* – `{'2017-01-01','2020-07-15'}` – `["2020-02-01","2020-04-30"]`
City	string character array	Abbreviation of the target city or `'rto'` for the entire market. *Default* – `'rto'`.
Others		See Section 6.5.11.

6.5.7 `plot_mobility`

Function Definition

```
function p=plot_mobility(market,varargin)
```

Function `plot_mobility` plots the appointed mobility curve of one of the seven cities. Like `plot`, it returns the line object of the curve for further modification. This function is based on MATLAB function `plot`.

By default, this function plots the mobility data of the retail sector in the specified city from 2019-12-30 to 2020-11-22, with no resampling. Additional name-value pair arguments are available for modifications. See Table 6-5 for a full list of arguments.

Table 6-5: Plot Mobility Options

Name	Type	Description
DateRange	datetime array string array cell array of character vectors	Range of time. *Default* – `{'2017-01-01','2020-07-15'}`. *Example* – `{'2017-01-01','2020-07-15'}` – `["2020-02-01","2020-04-30"]`
City	string character array	Abbreviation of the target city. *Default* – Corresponding city of the specified market.
Sector	string character vectors	The sector to be plotted. Can be assigned as `'Retail'`, `'Restaurant_Recreaction'` or `'Grocery_Pharmacy'`. *Default* – `'Retail'`.
ResampleBin	string character vectors	Binning scheme for resampling. *Default* – `'none'`.
ResampleMethod	string character vectors	Computation method for resampling. *Default* – `'mean'`.
Others		See Section 6.5.9.

6.5.8 plot_area

Function Definition

```
function ar=plot_area(t,varargin)
```

Function `plot_area` is the subfunction of `plot_generation_mix`. As the modification of MATLAB function `area`, several default settings are altered for appearance. The first column of table `t` is appointed as labels of the x-axis, and all the other columns contain data to be plotted. See Table 6-6 for a full list of arguments.

Table 6-6: Plot Area Options

Name	Type	Description
EdgeColor	RGB triplet hexadecimal color code ...	Color of the edges. *Default* – `'none'`.

ColorMap	table
	structure array
	string array
	cell array of character vectors
		Colors of all the areas in the plot.
		Default –
		`["coal","gas","oil","nuclear","hydro",`
		`"wind","solar","other","import";`
		`"#a67a6d","#f69850","#b35711","#3f9b98",`
		`"#a0d8f1","#73c698","#ffbd4a",`
		`"#b4b4b4","white"]`

6.5.9 plot_line_chart

Function Definition

```
function p=plot_line_chart(t,varargin)
```

Function `plot_line_chart` is the subfunction of `plot_renewable_share`, `plot_demand`, `plot_price` and `plot_mobility`. As the modification of MATLAB function `plot`, several default settings are altered for appearance. The first column of table `t` is appointed as labels of the x-axis, and the second column contains data to be plotted. See Table 6-7 for a full list of arguments.

Table 6-7: Plot Line Chart Options

Name	Type	Description
DisplayName	string character array	Name (legend) to be displayed on the graph. *Default* – Name of the table column.
Color	RGB triplet hexadecimal color code ...	Color of the line. *Default* – `'#0072BD'`.
LineWidth	positive numeric value	Width of the line. *Default* – `1`.
LineStyle	character value	Style of the line. *Default* – `'-'`.
Marker	character value	Style of the marker. *Default* – `'none'`.

6.5.10 `plot_quantile`

Function Definition

```
function gr_handles=plot_quantile(t,varargin)
```

Function `plot_quantile` is the subfunction of `plot_daily_demand_curve`. The intervals are represented with areas of different brightness, indicating the distribution. The first column of table `t` is appointed as labels of the x-axis, and the rest columns contain quantiles to be plotted. See Table 6-8 for a full list of arguments.

Table 6-8: Plot Quantile Options

Name	Type	Description
Color	RGB triplet hexadecimal color code ...	Color theme of the plot. *Default* – `'#0072BD'`.
Alpha	numeric vector	Face transparency of the objects. *Default* – `[0.1, 0.25]`.
LineWidth	positive value	Width of the central line (if exists). *Default* – `1`.

6.5.11 `plot_histogram`

Function Definition

```
function h=plot_histogram(t,varargin)
```

Function `plot_histogram` is the subfunction of `plot_price_distribution`. As the modification of MATLAB function `histogram`, several default settings are altered for appearance. The first column of table `t` is appointed as labels of the x-axis, and the second column contains data to be plotted. See Table 6-9 for a full list of arguments.

Table 6-9: Plot Histogram Options

Name	Type	Description
DisplayName	string character array	Name (legend) to be displayed on the graph. *Default* – Name of the table column.
Normalization	string character array	Method for data normalization. Can be assigned as `'count'`, `'probability'`, `'countdensity'`, `'pdf'`, `'cumcount'` or `'cdf'`. *Default* – `'pdf'`.
EdgeColor	RGB triplet hexadecimal color code ...	Color of the edges. *Default* – `'none'`.
NumBins	positive integer	Number of bins. *Default* – `50`.

6.6 Examples

In this section, we introduce the usage of the functions above with several examples. It is recommended to use feature-oriented high-level functions (see A.2), and the following explanations and instructions are designed for them.

6.6.1 Basic Usage

All high-level functions mentioned above have similar input-output forms and can be used similarly. To be specific, the first input argument (market name) is compulsory, and all the other input arguments are optional. The market name is specified as one of the seven terms:

`'caiso' | 'ercot' | 'isone' | 'miso' | 'nyiso' | 'pjm' | 'spp'`

All the other input arguments should be in the form of **name-value pairs** for the function to correctly recognize.

Command Window

```
plot_demand(market,'argument_name1',argument_value1,'argument_name2',
    argument_value2,...);
```

The output argument is the graphic object(s) of the plot, just like `plot`, `area` and `histogram`. The number of the object depends on the function type. For `plot_renewable_share`, `plot_demand`, `plot_price` and `plot_mobility`, the function returns only one line object. For `plot_generation_mix`, the number of area objects returned is equal to the number of fuel types. For `plot_price_distribution`, the function returns only one histogram object. For `plot_daily_demand_curve`, the function returns all the area objects and the line object (if exists) of the plot. They can be utilized to modify the plot after calling the function, as illustrated in Section 6.6.2.

As long as the market name is given, a basic version of the desired plot can be simply obtained. For example, to plot the price distribution histogram of ISONE, just enter

Command Window

```
>> plot_price_distribution('isone');
```

The result is displayed in Figure 6-3. By default, the price distribution histogram of ISONE from 2017-01-01 to 2020-07-15 is plotted. If the user wants to change the date range, parameter `DateRange` should be specified. In addition, some of the functions provide an option for resampling data, making them more flexible to use. For example, to plot the average price of every month in 2019, the user may enter

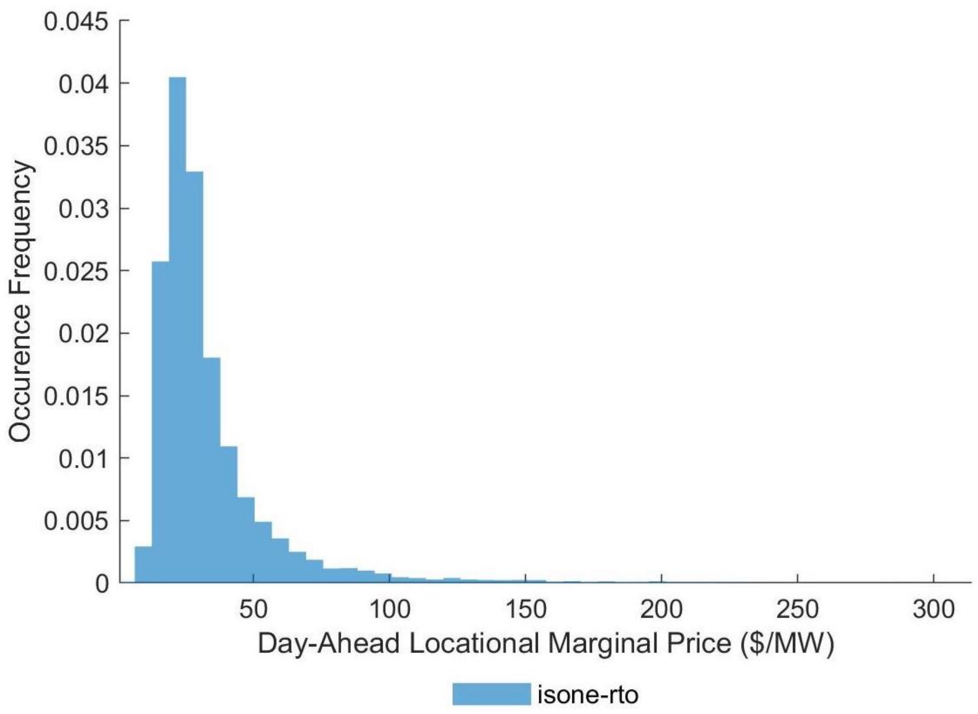

Figure 6-3: Example: Price Distribution of ISONE

Command Window

```
>> plot_price('nyiso','DateRange',{'2019-01-01','2019-12-31'},'ResampleBin
    ','month','ResampleMethod','mean');
```

The result is displayed in Figure 6-4, where the desired price trend is plotted.

Like `plot`, all the functions can be used more than once to plot multiple curves in one figure. For example, to plot the demand curves of NYISO in 2018, 2019 and 2020 with date alignment, first calculate the corresponding date range:

Command Window

```
>> dr_2020=datetime('2020-2-1'):datetime('2020-4-30');
>> dr_2019=dr_2020-calyears(1);
>> dr_2018=dr_2019-calyears(1);
```

Thus the desired date ranges for 2018, 2019 and 2020 are `dr_2018`, `dr_2019` and `dr_2020` correspondingly. Now plot the demand curves:

Command Window

```
>> plot_demand('nyiso','DateRange',dr_2018,'DisplayName','2018','LineStyle'
    ,':','Color','#939598');
>> plot_demand('nyiso','DateRange',dr_2019,'DisplayName','2019','LineStyle'
    ,'--','Color','#D95319');
>> plot_demand('nyiso','DateRange',dr_2020,'DisplayName','2020','Color','
    #0072BD','LineWidth',2.5);
```

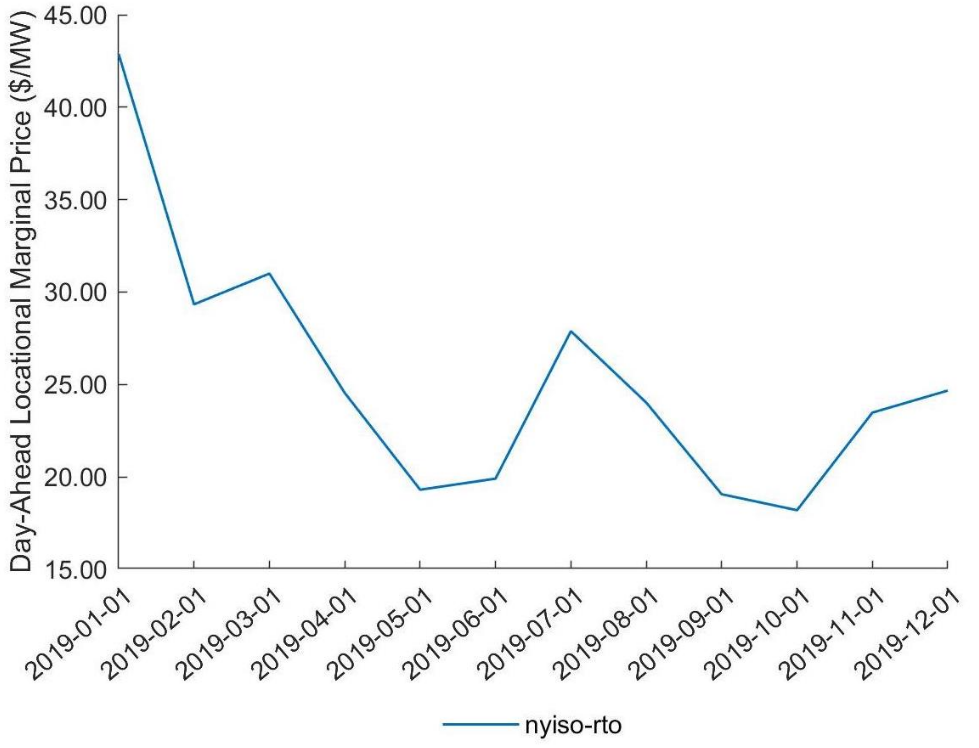

Figure 6-4: Example: Average Price of NYISO

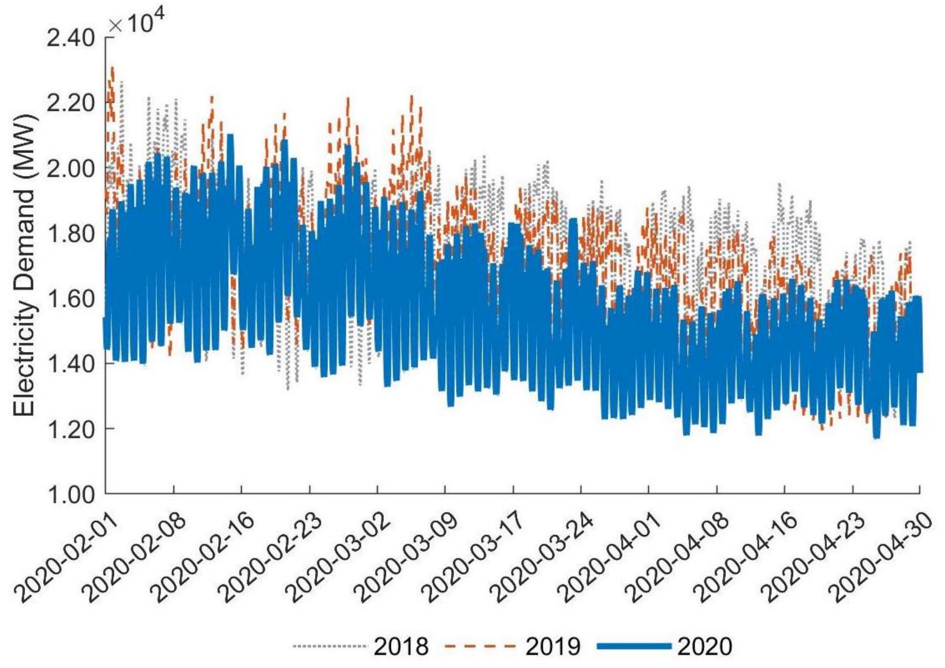

Figure 6-5: Example: Demand Curves of NYISO

The result is displayed in Figure 6-5, where the desired demand curves are plotted. In the example above, several properties of the lines are customized for better recognition. In Section 6.6.2, the modification of plots is explored further.

6.6.2 Customize Plot Properties

Like MATLAB, there are mainly two approaches in CoVEMDA to modifying a plot: before and after calling the plotting function. The former way refers to passing arguments in the form of **name-value pairs**, and the latter one refers to changing **property values** of the chart. They each have their own strengths, and both of them are commonly used to beautify a plot.

In the last example in Section 6.6.1, several properties (label, color and line width) of the three lines are modified by passing name-value pair arguments to the function. In this way, the user is able to integrate multiple modifications in one line of code, making it concise and simple. For example, to display markers at data points, the user may enter

Command Window

```
>> plot_price('nyiso','DateRange',{'2019-01-01','2019-12-31'},'ResampleBin'
    ,'month','ResampleMethod','mean','Marker','o');
```

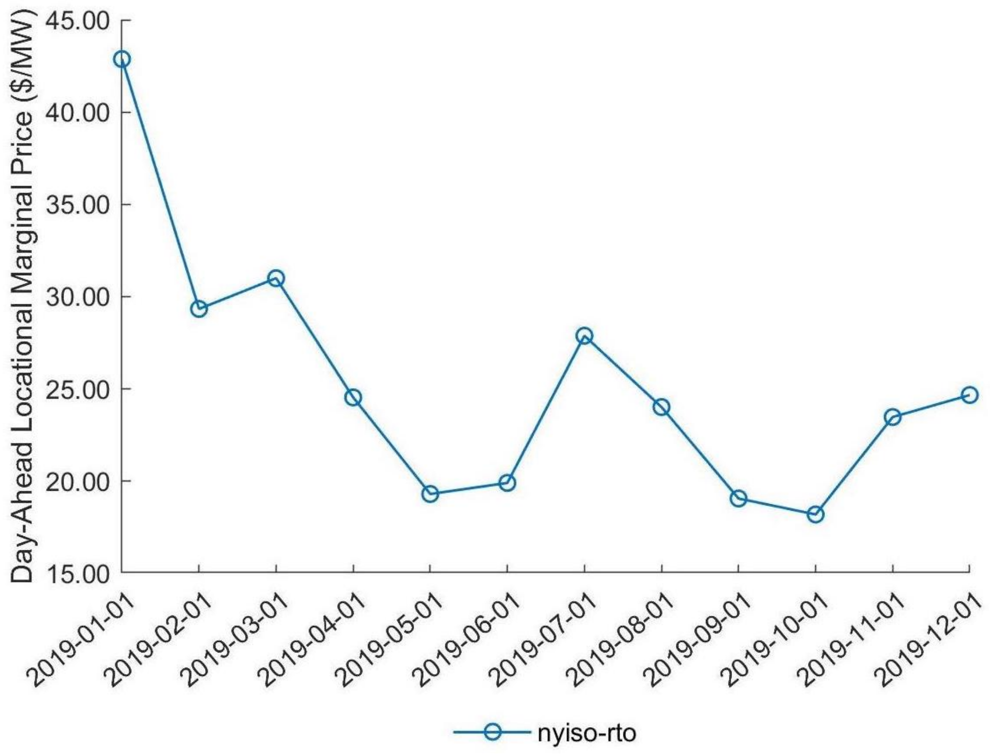

Figure 6-6: Example: Average Price of NYISO (with Markers)

The result is displayed in Figure 6-6, where all the data points are marked with a symbol 'o'. Any other marker symbol can be used in the same way, as long as it can be recognized by `plot`.

However, it is difficult to obtain detailed control of the plot by merely passing arguments to the functions, which is why sometimes we also need to change property values after plotting the figure. For example, to change the default display range of y-axis, the user may enter:

```
Command Window
>> plot_daily_demand_curve('nyiso');
>> ax=gca;
fx >> ax.YLim=[0,4];
```

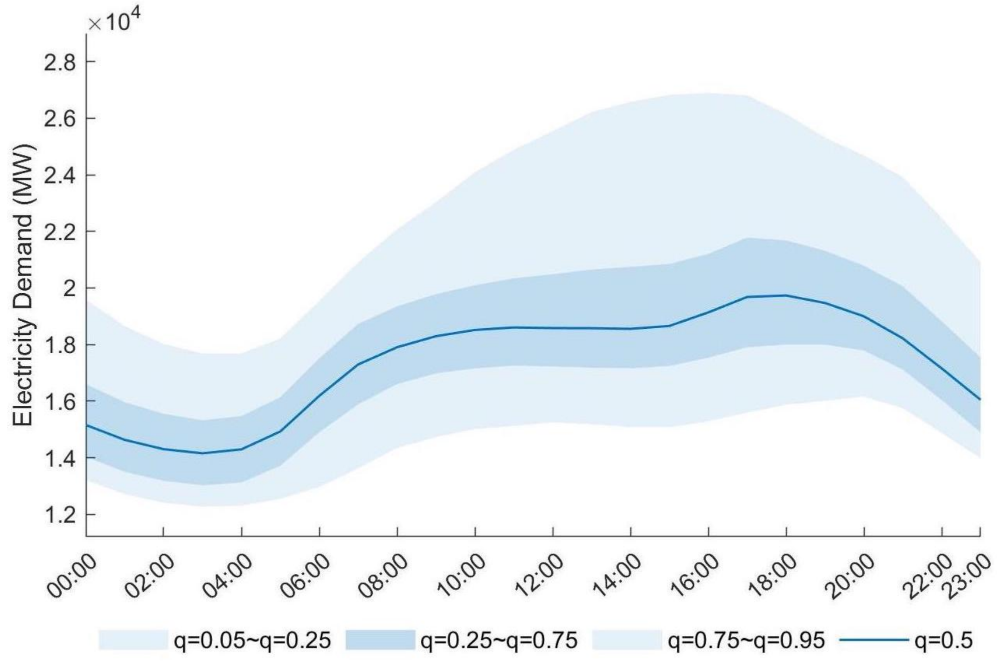

Figure 6-7: Example: Daily Demand Curve of NYISO (Before Changing range of y-axis)

The result is displayed in Figure 6-7 and 6-8, where the range of y-axis is enlarged. As the modified property belongs to the axes of the figure rather than area or line objects, the command `ax = gca` is utilized to access the axes. Also, `ylim([0, 4])` (using the function `ylim`) has the same effect as `ax.YLim = [0, 4]` in this case.

Under some circumstances, there may exist more than one object in the graph, which is common when using functions concerning `area`. For example, to highlight renewable energy in a generation mix graph, the user may enter

```
Command Window
>> ar=plot_generation_mix('spp');
>> ar{1}.FaceColor='#868686';
>> ar{2}.FaceColor='#acacac';
>> ar{4}.FaceColor='#7f7f7f';
fx >> ar{5}.FaceColor='#6b6b6b';
```

The result is displayed in Figure 6-9 and 6-10, where the color of all fuel types except hydro, solar and wind are changed to gray after the graph is plotted. Therefore, the renewables are visually highlighted.

Such modifications are obtained by changing property values of certain objects using dot notation. For a list of properties, see `Axes Properties`, `Line Properties`, `Area Properties` and `Histogram Properties`.

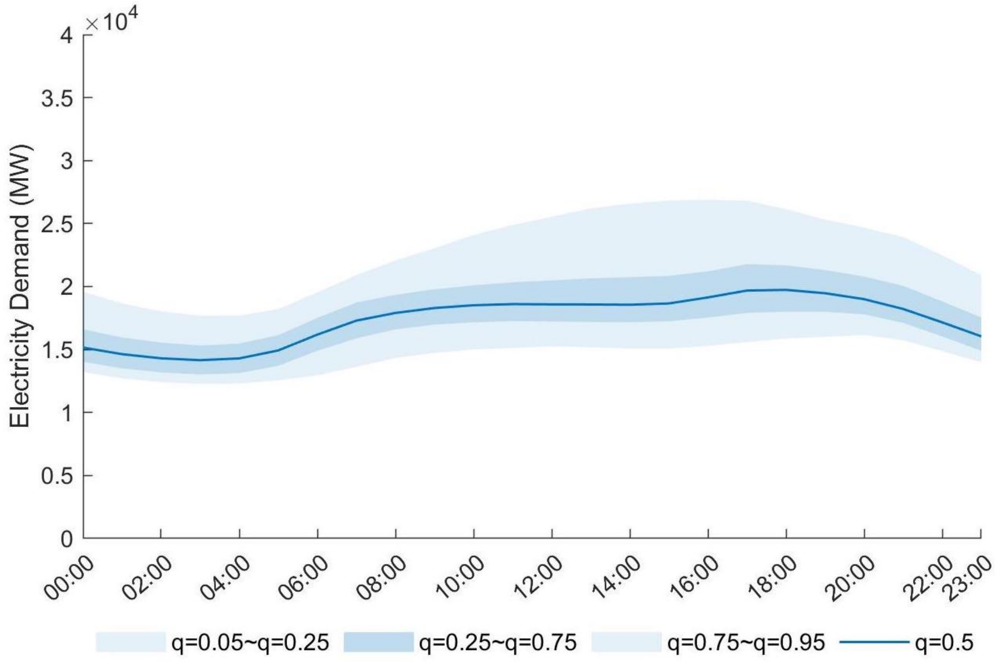

Figure 6-8: Example: Daily Demand Curve of NYISO (After Changing range of y-axis)

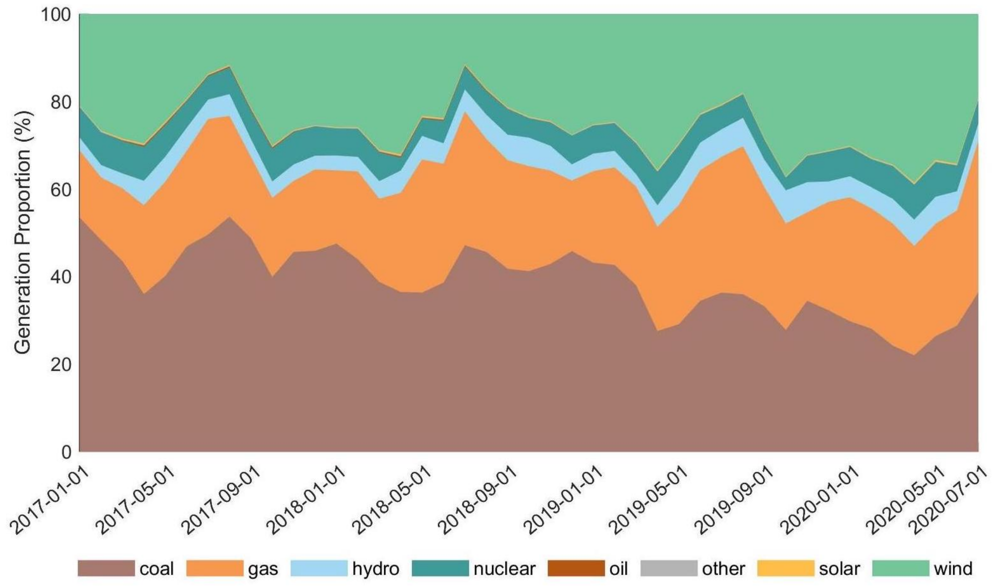

Figure 6-9: Example: Generation Mix of SPP (Before Highlighting the Renewables)

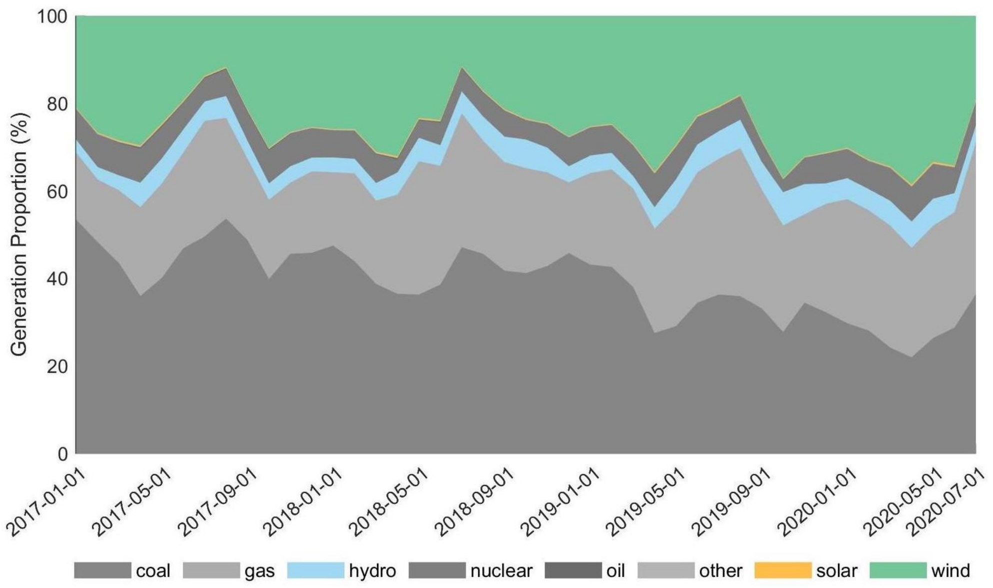

Figure 6-10: Example: Generation Mix of SPP (After Highlighting the Renewables)

7 Acknowledgments

The CoVEMDA Project is supported in part by the Student Research Training Program of Tsinghua University. We would like to acknowledge all experts for their suggestions during Tsinghua University Challenge Cup Competition. We are also grateful for the support of EILAB from Department of Electrical Engineering, Tsinghua University.

Appendix A CoVEMDA Files and Functions

A.1 Directory Layout

In CoVEMDA, the root directory contains four folders and two .m files. The two files are `install.m` and `uninstall.m` for installing and uninstalling of the toolbox, and folder `high_level`, `low_level` and `basic_operations` contain scripts of all other CoVEMDA functions.

Folder `data_temp` contains the temporary files of the toolbox, with two subfolders named `backcast` and `COVID-EMDA`. Folder `COVID-EMDA` contains .csv files obtained from the COVID-EMDA+ data hub, and folder `backcast\supplementary` contains supplementary files for ensemble backcast model (`gdp.csv`, etc.). Folder `backcast\models` contain pre-trained backcast models and their summary files, sorted in the following format:

```
models\<market_name>_<city_name_or_rto>\Model_<timestamp>\Models.mat
models\<market_name>_<city_name_or_rto>\Model_<timestamp>\Summary.csv
```

A.2 Functions

Apart from `install.m` and `uninstall.m` which are found in the root directory, CoVEMDA functions are typically divided into three levels: high level, low level and basic operations, which are also the corresponding folder names. All the high-level and low-level functions are illustrated above at the end of Section 6, Section 4 and Section 5.

Table A-1: CoVEMDA High-level Functions

Name	Description
`cal_abnormal_price_index`	calculates the abnormal price index in Section 4.3
`cal_baseline`	estimates the load baseline with three methods
`cal_distribution_diff`	calculates the Wasserstein difference of price distribution
`cal_excess_change_rate`	calculates the excess change rate of renewables in Section 4.5
`OLS`	gives a solution for the OLS regression model in Section 5.3.1
`VAR`	gives a solution for the VAR regression model in Section 5.3.2
`plot_baseline`	plots the load baseline with different methods
`plot_daily_demand_curve`	plots the demand curve with variation in one day
`plot_demand`	plots the demand curve
`plot_generation_mix`	plots the generation mix
`plot_mobility`	plots the mobility curve
`plot_price`	plots the price curve
`plot_price_distribution`	plots the price distribution histogram
`plot_renewable_share`	plots the renewable share curve

Table A-2: CoVEMDA Low-level Functions

Name	Description
`backcast_import_data`	imports data for ensemble backcast model
`backcast_scan_prediction`	calculates output of the backcast model for corresponding input
`backcast_scan_training`	trains the neural networks for the backcast model
`backcast_test_model`	selects the best neural networks to form the backcast model
`cal_baseline_by_backcast`	calculates load baseline by ensemble backcast model
`cal_baseline_by_dates`	calculates load baseline by date alignment
`cal_baseline_by_week_index`	calculates load baseline by week alignment
`plot_area`	plots filled areas
`plot_histogram`	plots histogram
`plot_line_chart`	plots lines
`plot_quantile`	plots data with variation

Table A-3: CoVEMDA Basic-operation Functions

Name	Description
`basic_backcast_import_models`	imports pre-trained backcast models for baseline estimation
`basic_backcast_save_file`	saves trained models and summary files
`basic_backcast_split_train_and_test`	splits training-validation set and testing set for each backcast model
`basic_backcast_split_train_and_valid`	splits training set and validation set for each network
`basic_backcast_train_and_validate`	trains and validates each network
`basic_calc`	basic four operations for tables
`basic_filter`	applies filter to table
`basic_fitdist`	fits probability distribution for table
`basic_gen_root_dir`	generates the root directory of CoVEMDA
`basic_group`	groups the table by non-date columns
`basic_max`	calculates the maximum value of all columns or rows
`basic_mean`	calculates the average value of all columns or rows
`basic_min`	calculates the minimum value of all columns or rows
`basic_plot_preprocess_table`	resamples and groups the table for plotting
`basic_quantile`	calculates the quantile of columns or rows
`basic_read`	reads files from COVID-EMDA+
`basic_resample`	resamples the table by date
`basic_sum`	sums up all columns or rows
`basic_exchange_columns`	exchange two columns in a table

References

[1] W. H. Organization, "Coronavirus disease (covid-19) pandemic," https://www.who.int/emergencies/diseases/novel-coronavirus-2019, 2021, retrieved June 8, 2021.

[2] K. T. Gillingham, C. R. Knittel, J. Li, M. Ovaere, and M. Reguant, "The short-run and long-run effects of covid-19 on energy and the environment," *Joule*, vol. 4, no. 7, pp. 1337–1341, 2020.

[3] H. Zhong, Z. Tan, Y. He, L. Xie, and C. Kang, "Implications of covid-19 for the electricity industry: A comprehensive review," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 3, pp. 489–495, 2020.

[4] R. M. Elavarasan, G. Shafiullah, K. Raju, V. Mudgal, M. T. Arif, T. Jamal, S. Subramanian, V. S. Balaguru, K. Reddy, and U. Subramaniam, "Covid-19: Impact analysis and recommendations for power sector operation," *Applied energy*, vol. 279, p. 115739, 2020.

[5] X. Xu and C.-f. Chen, "Energy efficiency and energy justice for us low-income households: An analysis of multifaceted challenges and potential," *Energy Policy*, vol. 128, pp. 763–774, 2019.

[6] G. Ruan, D. Wu, X. Zheng, H. Zhong, C. Kang, M. A. Dahleh, S. Sivaranjani, and L. Xie, "A cross-domain approach to analyzing the short-run impact of covid-19 on the us electricity sector," *Joule*, vol. 4, no. 11, pp. 2322–2337, 2020.

[7] G. Ruan, J. Wu, H. Zhong, Q. Xia, and L. Xie, "Quantitative assessment of us bulk power systems and market operations during the covid-19 pandemic," *Applied Energy*, vol. 286, p. 116354, 2021.

[8] T. K. Wijaya, M. Vasirani, and K. Aberer, "When bias matters: An economic assessment of demand response baselines for residential customers," *IEEE Transactions on Smart Grid*, vol. 5, no. 4, pp. 1755–1763, 2014.

[9] S. Mohajeryami, M. Doostan, A. Asadinejad, and P. Schwarz, "Error analysis of customer baseline load (cbl) calculation methods for residential customers," *IEEE Transactions on Industry Applications*, vol. 53, no. 1, pp. 5–14, 2016.

[10] L. Gosink, K. Bensema, T. Pulsipher, H. Obermaier, M. Henry, H. Childs, and K. I. Joy, "Characterizing and visualizing predictive uncertainty in numerical ensembles through bayesian model averaging," *IEEE transactions on visualization and computer graphics*, vol. 19, no. 12, pp. 2703–2712, 2013.

[11] J. A. Carrillo and G. Toscani, "Wasserstein metric and large–time asymptotics of nonlinear diffusion equations," in *New Trends in Mathematical Physics: In Honour of the Salvatore Rionero 70th Birthday*. World Scientific, 2004, pp. 234–244.

[12] L. V. Kantorovich, "On the translocation of masses," *Journal of mathematical sciences*, vol. 133, no. 4, pp. 1381–1382, 2006.

[13] A. Rzhetsky and M. Nei, "Statistical properties of the ordinary least-squares, generalized least-squares, and minimum-evolution methods of phylogenetic inference," *Journal of molecular evolution*, vol. 35, no. 4, pp. 367–375, 1992.

[14] C. A. Sims, "Macroeconomics and reality," *Econometrica: journal of the Econometric Society*, pp. 1–48, 1980.

[15] J. H. Stock and M. W. Watson, "Vector autoregressions," *Journal of Economic perspectives*, vol. 15, no. 4, pp. 101–115, 2001.

[16] Y. Liu, M. C. Roberts, and R. Sioshansi, "A vector autoregression weather model for electricity supply and demand modeling," *Journal of Modern Power Systems and Clean Energy*, vol. 6, no. 4, pp. 763–776, 2018.

[17] Y.-W. Cheung and K. S. Lai, "Practitioners corner: Lag order and critical values of a modified dickey-fuller test," *Oxford Bulletin of Economics and Statistics*, vol. 57, no. 3, pp. 411–419, 1995.

[18] D. B. Rubin, "Basic concepts of statistical inference for causal effects in experiments and observational studies," *Course material in Quantitative Reasoning*, vol. 33, 2003.

[19] "Deep learning toolbox user's guide," *The Mathworks, Inc.*, retrieved June 23, 2021 from https://www.mathworks.com/help/pdf_doc/deeplearning/nnet_ug.pdf.

[20] "Statistics and machine learning toolbox user's guide," *The MathWorks, Inc.*, retrieved June 23, 2021 from https://www.mathworks.com/help/pdf_doc/stats/stats.pdf.