1. Overview

Electronic companion of the paper: *Data-Driven Reliable Facility Location Design* by

- Hao Shen: School of Business, Renmin University of China, Beijing, China, shenhao@rmbs.ruc.edu.cn
- Mengying Xue (Corresopnding Author): School of Management, Fudan University, Shanghai 200433, China, mengying.xue@gmail.com
- Zuo-Jun Max Shen: Faculty of Engineering; Faculty of Business and Economics, The University of Hong Kong, Hong Kong, China maxshen@hku.hk

If you have any questions about how to use the code and generate the results, please contact the corresponding author.

This code was tested on:

- Ubuntu server equipped with 20 processors and 40G RAM.
- Gurobi 9.5 with Python API

Note: Please note that different versions of Gruobi and different platform may lead to small bias of the generated results.

2. Folders and Related Scripts

- `Data/` : Network data from [Synder and Daskin (2005)](Larry Snyder » Data Sets for “Reliability Models for Facility Location: The Expected Failure Cost Case”), processed weather data from NOAA from 1950 to 2021 in json file, [source] (Storm Events Database | National Centers for Environmental InformationStorm Events Database | National Centers for Environmental Information; folders for simulation data
  - `RawDataNoCov/` : stores the generated simulation training and testing data based on randomly generated distributions in the no-covariate case (Section 5 of the paper). The data format is .csv.
  - `RawDataNoCovFileName/` : stores the name lists for the simulation traning and testing data with no covariates. The names are recorded in .txt files.
  - `RawData/` : stores the generated simulation training and testing data based on randomly generated distributions in the multi-covariate case (Section 6 of the paper).
  - `RawDataFileName/` : stores the name lists for the simulation training and testing data with covariates. The names are recorded in .txt files.
  - `RawDataDistribution/` : stores the randomly generated distributions on supplies and demands. The data format is .pkl. For the no covariate tests, the file is with the name "Node_%d-mu_%f.pkl", with the number of nodes in the network and the demand means (High-1.6 or Low-0.4). For the tests with multi-covariates, the file is in the name "Node_%d-cov_%d.pkl", with the number of nodes and the number of covariates.
  - `RawDataStorm/` : stores the generated synthetic data sets for the case study in Section 6. Each generated file records the states of each of 49 locations in 12 months over a year, from 1997 to 2021.
  - `RawDataStormFileName/` : stores the name lists for the synthetic data sets for case study.
  - `Storm/` :
    - `disruption_49.json` : synthetic data file for the network with 49 nodes with disruption states using the data from NOAA. We refer readers to Section 6 in the manuscript to see the data processing of the weather data. Each row represents one historical record of weather hazard, and each column represents the state (disruption or not) of the 49 locations on the network. To generation this file, run `DisruptionState()` in `data_generate.py` .
    - `StormEvents_details_Begin_1950_End_2021.csv` : the aggregated data of the historical records downloaded from NOAA. Used for generating `disruption_49.json` .

    - `dataXXXUFLP.xls` : XXX in [49,50,100,150], location data in [Synder and Daskin (2005)](Reliability Models for Facility Location)
    - `UCFLDataXXX.txt` : XXX in [10,20,50,75,100,150], location data in Synder and Daskin (2005) and later processed by [Li et al. (2022)](A General Model and Efficient Algorithms for Reliable Facility Location Problem Under Uncertain Disruptions | INFORMS Journal on Computing)
- `DataRelated/` : Python code for preprocessing raw data and generating synthetic data used in numerical studies.
    - ``DataGenerateNoCov.py` :generates synthetic data for the numerical experiments in Section 5.
    - `DataGenerate.py` : loads the information from Synder and Daskin's data set, and generates synthetic data in the multi-covarate case for the numerical experiments in Section 6.2.
    - `DataGenerateStorm.py` : generates the monthly location states data based on the weather data from NOAA for the test in Section 6.3.
    - `DataProcess.py` : used for the data processing before the optimization in our PUB estimator.
    - `StormDataProcess.py` : processes the original weather data from NOAA. See the details in Section 6 in the paper.
- `MomentBased/` : Python code for marginal-moment method in [Lu et al. (2015)](Reliable Facility Location Design Under Uncertain Correlated Disruptions | Manufacturing & Service Operations Management) and cross moment method in [Li et al. (2022)](A General Model and Efficient Algorithms for Reliable Facility Location Problem Under Uncertain Disruptions | INFORMS Journal on Computing), denoted as MM, CM, or MM-COV, CM-COV in the paper.
    - `CrossMomentFunc.py` : functions for cross moment method
    - `SUPCrossMoment.py` : the constraint-generation optimization function for cross moment method. This original method was revised by us.
    - `MarginalMoment.py` : functions for marginal moment method, including the constraint-generation optimziation function,
    - `MomentData.py` : functions on obtaining the moment information from the data.
- `results/` : folder for average performances of all the tested methods, including PUB, WASS, MM, and CM in Section 5.1, PUB-COV, MM-COV, and CM-COV in Section 6.
    - `/reliability/` : detailed results for each data set
    - `/Storm/` : results for the case study
- `Wasserstein/` :Python code for optimizing the RFLP using type-\infty wasserstein DRO in [Xie (2020)](Tractable reformulations of two-stage distributionally robust linear programs over the type-∞ Wasserstein ball - ScienceDirectTractable reformulations of two-stage distributionally robust linear programs over the type-∞ Wasserstein ball - ScienceDirect), denoted as Wass in the paper.
    - `Wasserstein.py`
- `PUB/` : Python code for RFLP using PUB estimator proposed by Data-driven Reliable Faciltiy Location Problem, denoted as PUB or PUB-COV in the paper.
    - `SUPDataDriven.py` : constraint-generation algorithm
    - `SUPFunc.py` : functions on obtaining the cost, worst-case distributions
- `SampleAverage/` : Python code for RFLP using sample average approximation method.
    - `SAA.py` : optimizaiton function for SAA
    - `SAAFunc.py` functions on obtaining the cost in SAA
- `Reliability/` : Python codes for testing reliability and generating average performances of RFLP using PUB estimator, Wasserstein DRO, and moment based methods.
    - `Reliability_PUB.py` :functions on finding the conservative paramters for PUB estimators in in the paper Data-driven Reliable Faciltiy Location Design.
    - `Reliability_Wass.py` : functions on finding the conservative paramters for type-∞ Wasserstein DRO in [Xie (2020)] (Tractable reformulations of two-stage distributionally robust linear programs over the type-∞ Wasserstein ball - ScienceDirectTractable reformulations of two-stage distributionally robust linear programs over the type-∞ Wasserstein ball - ScienceDirect).

    - `test_reliability_moment.py` : generate the performances of four moment-based methods, including marginal moment with no covariates and with covariates, and cross-moment method with no covariates and with covariates, in the numerical experiments, including the objectives, out-of-sample costs, reliability, optimality gap, computational time.
    - `test_reliability_PUBNoCov.py` : generate the performances for the PUB estimator in the no-covariate case.
    - `test_reliability_PUB.py` : generate the performances for the PUB estimator in the multi-covariate case.
    - `test_reliability_SAA.py` : generate the performances for the sample average approximation method.
    - `test_reliability_Wass.py` : generate performances for the type-\infty average approximation method.
- `Utility/` : Python code for utility functions and constants used in the experiments
    - `Constants.py` : constants used in the experiments
    - `dist.py` : functions on calculating the distance between two locations
    - `Utils.py` : utility functions
- `Plot/` : Python code for the plots in the paper Data-driven Reliable Faciltiy Location Design.
    - `plot.py`
- `Results_Paper` : Original numeric results for the figures and tables in the paper, using the data in `Data/`.

Scripts and Modules

Run each of the following scripts, and the result file is in the format of `.csv` and will be stored in the folder `/result/`. The plots will be in the format of `.pdf` and will be stored in the folder `/plot/`.

- `data_generate.py` : generates synthetic data and case study data
- `run_Figure1_abc.py` : generates the numerical results for plotting Figure 1 (a) (b) (c) on page 21.
- `plot_Figure1_abc.py` : generates Figure 1 (a) (b) (c) after finishing running `run_Figure1_abc.py`.
- `run_Figure1_def.py` : generates the numerical results for plotting Figure 1 (d) (e) (f) on page 21.
- `plot_Figure1_def.py` : generates Figure 1 (d) (e) (f) after finishing running `run_Figure1_def.py`.
- `run_Figure2_abc.py` : generates the numerical results for plotting Figure 2 (a) (b) (c) on page 22.
- `plot_Figure2_abc.py` : generates Figure 2 (a) (b) (c) after finishing running `run_Figure2_abc.py`.
- `run_Figure2_def.py` : generates the numerical results for plotting Figure 2 (d) (e) (f) on page 22.
- `plot_Figure2_def.py` : generates Figure 2 (d) (e) (f) after finishing running `run_Figure2_def.py`.
- `run_Efficiency.py` : generates the numerical results for computational efficiency tests in Section 5.2.
- `run_Figure4.py` : generates the numerical results for plotting Figure 4 on page 27.
- `plol_Figure4.py` : generates Figure 4 after finishing running `run_Figure4.py`.
- `run_Case_Study.py` : generates the numerical results for the case study.
- `plot_Case_Study.py` : generates FIgure 5 and the numbers in Table 1 after finishing running `run_Case_Study.py`.
- `run_SAA_NoCov.py` : generates the approximated true optimum for the tests in Figure 1 and Figure 2 in Section 5.1.
- `run_SAA_Cov.py` : generates the approximated true optimum for the tests in Figure 4 in Section 6.2.

The following scripts are designed for running the single method in each test. You can skip these if you are not interested.

- `run_PUB_NoCov.py` generates performances for PUB estimators in data with no covariate (results for Section 5.1 and 5.2)
- `run_PUB_Cov.py` :generates performances for PUB estimators in data with covariates (results for Section 6.2)
- `run_moment_NoCov.py` : generates test results for marginal momemt and cross moment methods in the data with no covariate (results for Section 5.1 and 5.2)
- `run_moment_Cov.py` : generates test results for the marginal momemt with no cov and cov, and cross moment method with no cov and cov in the data with multi-covariates (results for Section 6.2)
- `run_wass.py` : generates test results type-∞ wasserstein DRO method.

- `run_PUB_Storm.py` : generates the performance of PUB estimator with covariates in the case study using the weather data from NOAA (results in Section 6.3)
- `run_moment_storm.py` : generates the performance of marginal and cross moment based methods with covariates in the case study using the weather data from NOAA (results in Section 6.3)

3. Implementation

This section provides step-by-step instructions for the all the numerical results presented in the paper.

Key Instructions

1. Please read and follow these steps carefully for a successful implementation.

2. **The simulation data we used for the paper has been put in the folder `/Data/`**. All experiments will access data files from this location. For details on data generation, please refer to Section **3.8 Generating Simulation Data**.

Implementation steps

Section 5.1.1, Figure 1 (a) (b) (c) in page 21: Run the following three steps:

1. **Set Line 10 of `run_SAA_NoCov.py` as**

```
mu_coeff = 1.6
```

and run "run_SAA_NoCov.py".

Note:

(i) The input for this step is `/Data/RawDataNoCov/Real_Node_10-mu_1.600000-truncate_1.csv`, and the output for this step will be saved in `/result/SAA_node10_Mu_1.600000_Truncate_1.csv`.

2. **Set Line 17 of Python Script `run_Figure1_abc.py` as:**

```
NumDataSet = [10,25,50,75,100,250,500,750,1000]
```

and run `run_Figure1_abc.py`.

Note:

(i) The input for this step is `Data/RawDataNoCovFileName/Train_Node_10-mu_1.600000-truncate_1.txt` (the list of filenames of the simulated data files) and `Data/RawDataNoCov/Train_Node_10-mu_1.600000-truncate_1-Simulate_X.csv` for X from 0 to 99, and `Data/RawDataNoCov/Test_Node_10-mu_1.600000-truncate_1-Length_10000.csv`, and `Data/UCFLData10.txt`. The output for this step is put in `/result/`, with the name `Reliability_KolNoCov_Node10_DataX_Cov1_Beta0.1000_Mu_1.600000_Truncate_1.csv` and `Reliability_Wasserstein_Node10_DataX_Cov1_Beta0.1000_Mu_1.600000_Truncate_1.csv` with X in NumDataSet.

(ii) In the python script `run_Figure1_abc.py`, we run 9 problem instances, indexed by variable "num_data" selected from array "NumDataSet", for each of the two methods (i.e., PUB, Wass), where the results for PUB is obtained by function *test_Reliability_PUB_NoCov()*, and the results for Wass is obtained by the function *test_Reliability_Wasserstein()*. For each problem instance and each method, we obtain three performance measures, namely, the "in-sample performance", the "out-of-sample performance", and the "reliablility", of this method.

(iii) If you run all the instances using the FOR loop as shown in `run_Figure1_abc.py`, the total computational time will be several days. However, all 9 instances of two functions can be run in parallel.

3. **Run `plot_Figure1_abc.py`**

Note

1. The numerical results we have generated for Figure 1(a) (b) (c) are stored in folder `/Reults_Paper/`. If you want to generate the exact figure in the paper, copy the contents in `/Results_Paper/` to `/result/`, and run this step.

2. There will be three generated figures in ".pdf" in the folder "/Plot/: The correspondence with Figure 1 (a),(b),(c) on Page 21 are:

Figure 1-(a): `In-sample Performance_PUB_Wass_Node10_NoCov_Case_H.pdf`

Figure 1-(b): `Out-of-sample Performance_PUB_Wass_Node10_NoCov_Case_H.pdf`

Figure 1-(c): `Reliability_PUB_Wass_Node10_NoCov_Case_H.pdf`

Section 5.1.1, Figure 1 (d) (e) (f): Run the following three steps:

1. **Set Line 10 of `run_SAA_NoCov.py` as**

```
mu_coeff = 0.4
```

and run `run_SAA_NoCov.py`

Note:

(i) The input for this step is `/Data/RawDataNoCov/Real_Node_10-mu_0.400000-truncate_1.csv`, and the output for this step will be saved in `/result/SAA_node10_Mu_0.400000_Truncate_1.csv`.

2. **Set Line 17 of Python Script `run_Figure1_def.py` as**

```
NumDataSet = [10,25,50,75,100,250,500,750,1000]
```

and run `run_Figure1_def.py`.

Note:

(i) The input for this step is `Data/RawDataNoCovFileName/Train_Node_10-mu_0.400000-truncate_1.txt` (the list of filenames of the simulated data files) and `Data/RawDataNoCov/Train_Node_10-mu_0.400000-truncate_1-Simulate_X.csv` for X from 0 to 99, and `Data/RawDataNoCov/Test_Node_10-mu_0.400000-truncate_1-Length_10000.csv`, , and `Data/UCFLData10.txt`. The output for this step is put in `/result/`, with the name

`"Reliability_KolNoCov_Node10_DataX_Cov1_Beta0.1000_Mu_0.400000_Truncate_1.csv` and `Reliability_Wasserstein_Node10_DataX_Cov1_Beta0.1000_Mu_0.400000_Truncate_1.csv` with X in NumDataSet.

(ii) In the python script `run_Figure1_def.py`, we run 9 problem instances, indexed by num_data selected from NumDataSet, for each of the three methods (i.e., PUB, Wass), where the results for PUB is obtained by function *test_Reliability_PUB_NoCov()*, and the results for Wass is obtained by the function *test_Reliability_Wasserstein()*. For each problem instance and each method, we obtain three performance measures, namely, the "in-sample performance", the "out-of-sample performance", and the "reliablility", of this method.

(iii) If you run all the instances using the FOR loop as shown in `run_Figure1_def.py`, the total computational time will be several days. However, all 9 instances of two functions can be run in parallel.

3. **Run `plot_Figure1_def.py`.**

Note

1. The numerical results we have generated for Figure 1(d) (e) (f) are copied in folder `/Reults_Paper/`. If you want to generate the exact figure in the paper, copy the contents in `/Reults_Paper/` to `/result/`, and run this step.

2. There will be three generated figures in ".pdf" in the folder **"/Plot/**: The correspondence with Figure 1 (d),(e),(f) on Page 21 are:

Figure 1-(d): `In-sample Performance_PUB_Wass_Node10_NoCov_Case_L.pdf`

Figure 1-(e): `Out-of-sample Performance_PUB_Wass_Node10_NoCov_Case_L.pdf`

Figure 1-(f): `Reliability_PUB_Wass_Node10_NoCov_Case_L.pdf`

Section 5.1.2 Figure 2 (a) (b) (c): Run the following three steps:

1. **Set Line 10 of `run_SAA_NoCov.py` as**

```
mu_coeff = 1.6
```

and run "run_SAA_NoCov.py".

Note:

(i) The input for this step is `/Data/RawDataNoCov/Real_Node_10-mu_1.600000-truncate_1.csv`, and the output for this step will be saved in `/result/SAA_node10_Mu_1.600000_Truncate_1.csv`.

(ii) If you have done the steps for Section 5.1.1 Figure 1(a) (b) (b), you can skip this step.

2. **Set Line 12 of Python Script `run_Figure2_abc.py` as:**

```
NumDataSet = [10,25,50,75,100,250,500,750,1000]
```

and run `run_Figure2_abc.py".

Note

(i) The input for this step is `Data/RawDataNoCovFileName/Train_Node_10-mu_1.600000-truncate_1.txt` (the list of filenames of the simulated data files) and `Data/RawDataNoCov/Train_Node_10-mu_1.600000-truncate_1-Simulate_X.csv` for X from 0 to 99, `Data/RawDataNoCov/Test_Node_10-mu_1.600000-truncate_1-Length_10000.csv`, and `Data/UCFLData10.txt`.The output for this step is put in `/result/`, with the name `"Reliability_KolNoCov_Node10_DataX_Cov1_Beta0.1000_Mu_1.600000_Truncate_1.csv` and `Reliability_moment_Node10_DataX_Cov1_Beta0.1000_Mu_1.600000_Truncate_1.csv` with X in NumDataSet.

(ii) In the python script `run_Figure2_abc.py`, we run 9 problem instances, indexed by num_data selected from NumDataSet, for each of the three methods (i.e., PUB, MM, CM), where the results for PUB is obtained by function *test_Reliability_PUB_NoCov()*, and the results for both MM and CM are obtained by the function *test_Reliability_mm_NoCov()*. For each problem instance and each method, we obtain three performance measures, namely, the "in-sample performance", the "out-of-sample performance", and the "reliablility", of this method.

(iii) If you run all the instances using the FOR loop as shown in `run_Figure2_abc.py`, the total computational time will be several days. However, all 9 instances of the two functions could be run in parallel.

(iv) If the python script `run_Figure1_abc.py` has been run, then the code on Line 19.

```
 test_Reliability_PUB_NoCov(train_data_lst, test_data, info, num_node, num_data, 1, beta_, mu_coeff,
truncate)
```

can be commented out. Since the results for "PUB_NoCov" has been generated when runing the experiments for Section 5.1.1.

3. **Run `plot_Figure2_abc.py`**,

Note

1. The numerical results we have generated for Figure 2 (a) (b) (c) are copied in folder `/Reults_Paper/`. If you want to generate the exact figure in the paper, copy the contents in `/Reults_Paper/` to `/result/`, and run this step.

2. There will be three generated figures in ".pdf" in the folder **"/Plot/**: The correspondence with Figure 2 (a),(b),(c) on Page 22 are:

Figure 2-(a): `"In-sample Performance_PUB_Moment_Node10_NoCov_Case_H.pdf`

Figure 2-(b): `Out-of-sample Performance_PUB_Moment_Node10_NoCov_Case_H.pdf`

Figure 2-(c): `Reliability_PUB_Moment_Node10_NoCov_Case_H.pdf`

Section 5.1.2 Figure 2 (d) (e) (f): Run the following three steps:

1. **Set Line 10 of `run_SAA_NoCov.py` as**

```
mu_coeff = 0.4
```

and run `run_SAA_NoCov.py`

Note:

(i) The input for this step is `/Data/RawDataNoCov/Real_Node_10-mu_0.400000-truncate_1.csv`, and the output for this step will be saved in `/result/SAA_node10_Mu_0.400000_Truncate_1.csv`.

(ii) If you have done the steps for Section 5.1.1 Figure 1(d) (e) (f), you can skip this step.

2. **Set Line 12 of Python Script `run_Figure2_def.py` as:**

```
NumDataSet = [10,25,50,75,100,250,500,750,1000]
```

and run `run_Figure2_def.py`.

Note

(i) The input for this step is `Data/RawDataNoCovFileName/Train_Node_10-mu_0.400000-truncate_1.txt` (the list of filenames of the simulated data files) and `Data/RawDataNoCov/Train_Node_10-mu_0.400000-truncate_1-Simulate_X.csv` for X from 0 to 99, `Data/RawDataNoCov/Test_Node_10-mu_0.400000-truncate_1-Length_10000.csv`, and `Data/UCFLData10.txt`.The output for this step is put in `/result/`, with the name `"Reliability_KolNoCov_Node10_DataX_Cov1_Beta0.1000_Mu_0.400000_Truncate_1.csv` and `Reliability_moment_Node10_DataX_Cov1_Beta0.1000_Mu_0.400000_Truncate_1.csv` with X in NumDataSet.

(ii) In the python script `run_Figure2_def.py`, we run 9 problem instances, indexed by num_data selected from NumDataSet, for each of the three methods (i.e., PUB, MM, CM), where the results for PUB is obtained by function *test_Reliability_PUB_NoCov()*, and the results for both MM and CM are obtained by the function *test_Reliability_mm_NoCov()*. For each problem instance and each method, we obtain three performance measures, namely, the "in-sample performance", the "out-of-sample performance", and the "reliablility", of this method.

(iii) If you run all the instances using the FOR loop as shown in `run_Figure2_def.py`, the total computational time will be several days. However, all 9 instances of the two functions could be run in parallel.

(iv) If the python script `run_Figure1_def.py` has been run, then the code on Line 19,

```
 test_Reliability_PUB_NoCov(train_data_lst, test_data, info, num_node, num_data, 1, beta_, mu_coeff,
truncate)
```

can be commented out. Since the results for "PUB_NoCov" has been generated when runing the experiments for Section 5.1.1.

3. **Run `plot_Figure2_def.py`**

Note

1. The numerical results we have generated for Figure 2 (d) (e) (f) are copied in folder `/Reults_Paper/`. If you want to generate the exact figure in the paper, copy the contents in `/Reults_Paper/` to `/result/`, and run this step.

2. There will be three generated figures in ".pdf" in the folder **"/Plot/**: The correspondence with Figure 2 (d),(e),(f) on Page 22 are:

Figure 2-(d): `In-sample Performance_PUB_Moment_Node10_NoCov_Case_L.pdf`

Figure 2-(e): `Out-of-sample Performance_PUB_Moment_Node10_NoCov_Case_L.pdf`

Figure 2-(f): `Reliability_PUB_Moment_Node10_NoCov_Case_L.pdf`

Section 5.2 Computational Efficiency (Results shown in Table EC.1 in the appendix): Run the following step:

1. Run `run_Efficiency.py`. The settings are

```
num_node_lst = [10,20,50]
beta_ = 0.1
NumDataSet = [100,500,1000,5000,10000]
mu_coeff = 1.6
truncate = 1
```

Note

(i) The input for this step is `Data/RawDataNoCovFileName/Train_Node_Y-mu_1.600000-truncate_1.txt` (the list of filenames of the simulated data files) and `Data/RawDataNoCov/Train_Node_Y-mu_1.600000-truncate_1-Simulate_X.csv` for X from 0 to 99 and Y in *num_node_lst*, `Data/RawDataNoCov/Test_Node_10-mu_1.600000-truncate_1-Length_10000.csv` , and `Data/UCFLData10.txt`.

(ii) The generated results will be in the folder `/result/`, with the name

The files for the method **"PUB"** will be in the name as"
`Reliability_KolNoCov_NodeX_DataY_Cov1_Beta0.1000_Mu_1.600000_Truncate_1.csv`

The files for the method **"Wass"** will be in the name as"
`Reliability_Wasserstein_NodeX_DataY_Cov1_Beta0.1000_Mu_1.600000_Truncate_1.csv"`

The files for the method **"MM"** or **CM** will be in the name as
`Reliability_moment_NodeX_DataY_Cov1_Beta0.1000_Mu_1.600000_Truncate_1.csv`, with "**MM**" corresponds with the "MM_nocov" in the table, "**CM**" corresponds with the "CM_nocov" in the table,

where X is in *num_node_lst*, Y is in *NumDataSet*.

(iii) In the python script `run_Efficiency.py`, we run 12 problem instances, indexed by num_node selected from num_node_lst paired with num_data selected from NumDataSet, for each of the four methods (i.e., PUB, Wass, MM, CM), where the results for PUB is obtained by function *test_Reliability_PUB_NoCov()*, the results for Wass is generated from *test_Reliability_Wasserstein()* , and the results for both MM and CM are obtained by running the function *test_Reliability_mm_NoCov()*. For each problem instance and each method, we obtain four performance measures, namely, the "in-sample performance", the "out-of-sample performance", the "solution time" and "Gap", of this method.

(iv) In Table EC.1, "In-sample cost ($\times 10^5$)" corrseponds with the column "**in_sample_cost_avg**" of the result file (the number divided by 10,0000). "Out-of-sample cost ($\times 10^5$)" corresponds with the values in column "**out_of_sample_cost_avg** of the result file (the number divided by 10,0000), and the "Solution time (s)" corresponds with the values in the column **time_sol** of the result file, the "Gap (%)" correponds with **Gap** in each result file.

(v) If you run all the instances using the FOR loop as shown in `run_Efficiency.py` , the total computational time will be several weeks. However, all 12 instances of three functions could be run in parallel.

Section 6.2 Figure 4: Run the following three steps:

1. **Run `run_SAA_Cov.py`**

   Note:

   (i) The input for this step is `/Data/RawData/Real_Node_10-Cov_2.csv`, and the output for this step will be saved in `/result/SAA_node10_cov2.csv`.

2. **Set Line 14 of Python Script `run_Figure4.py` as:**

```
NumDataSet = [10,25,50,75,100,250,500,750,1000]
```

and run `run_Figure4.py`.

Note

(i) The input for this step is `Data/RawDataFileName/Train_Node_10-Cov_2.txt` (the list of filenames of the simulated data files) and `Data/RawData/Train_Node_10-Cov_2-Simulate_X.csv` for X from 0 to 99, `Data/RawDataNoCov/Test_Node_10-Cov_2-Length_10000.csv`, and `Data/UCFLData10.txt`.The output for this step is put in `/result/`, with the name `"Reliability_Kol_Node10_DataX_Cov2_Beta0.1000.csv` and `Reliability_moment_Node10_DataX_Cov2.csv` with X in NumDataSet.

(ii) In the python script `run_Figure4.py`, we run 9 problem instances, indexed by num_data selected from NumDataSet, for each of the three methods (i.e., PUB-COV, MM-COV, CM-COV), where the results for PUB-COV is obtained by function *test_Reliability_PUB()*, and the results for both MM-COV and CM-COV are obtained by the function *test_Reliability_mm()*. For each problem instance and each method, we obtain three performance measures, namely, the "in-sample performance", the "out-of-sample performance", and the "reliablility", of this method.

(iii) If you run all the instances using the FOR loop as shown in `run_Figure4.py`, the total computational time will be several days. However, all 9 instances of two functions could be run in parallel.

3. **Run `plot_Figure4.py`**

Note

1. The numerical results we have generated for Figure 4 are copied in folder `/Reults_Paper/`. If you want to generate the exact figure in the paper, copy the contents in `/Reults_Paper/` to `/result/`
2. There will be three generated figures in ".pdf" in the folder **"/Plot/**: The correspondence with Figure 4 (a),(b),(c) on Page 27 are:

Figure 4-(a): `In-sample Performance_PUB_Moment_Node10_Cov2.pdf`

Figure 4-(b): `Out-of-sample Performance_PUB_Moment_Node10_Cov2.pdf`

Figure 4-(c): `Reliability_PUB_Moment_Node10_Cov2.pdf`

Section 6.3 Case Study: Figure 5 and Table1: Run the following three steps:

1. **Change the setting in `/Utility/Constants.py`**

```
Num_reliability = 30
START_YEAR = 1996
END_YEAR = 2021
```

2. **Run `run_Case_Study.py`**

   Note:

(i) The input for this step is `Data/RawDataStormFileName/Train-TrainLength_1-TestLength_1-Node_49-Cov_2.txt` and `Data/RawDataStormFileName/Test-TrainLength_1-TestLength_1-Node_49-Cov_2` (the list of filenames of the simulated data files), and `Data/RawDataStorm/Train-TrainLength_1-TestLength_1-Node_49-Cov_2-Year_X.csv` for X from 1996 to 2020, `Data/RawDataStorm/Test-TrainLength_1-TestLength_1-Node_49-Cov_2-Year_X.csv` for X from 1996 to 2020, and `Data/data49UFLP.xls`. The output for this step is put in `/result/Storm`, with the name `Moment-Node_49-Cov_2-Beta_0.20.csv` (for MM-COV and CM-COV)and `Kol_cov-Node_49-Cov_2-Beta_0.20.csv` (for PUB-COV).

3. **Run `plot_Case_Study.py`**

   Note

   1. The numerical results we have generated for the case study are copied in folder `/Reults_Paper/Storm/`. If you want to generate the exact figure in the paper, copy the contents in `/Results_Paper/Storm/` to `/result/Storm/`

   Figure 5 in the paper corresponds with the file `Plot/Figure5.pdf`, the numbers in Table 3 will be printed out on screen.

For generating Simulation Data

1. Change the setting in Line 22 of **"/Utility/Constants.py"** as

2. ```
   Num_reliability = 50 ## number of loops when searching for the radius of r
   Num_Simulate = 100 ## number of data samples we use for the study
   ```

3. Run "data_generate.py". The generated data files will be in the folder `/Data/`, which will cover the original data files.

4. There exists some randomness in the generated data. Thus, if you regenerate a new set of simulated data, the figures and tables in the paper might not be exactly recovered, but the trends and patterns will hold. In particular, the advantage of **PUB** or **PUB-COV** over other methods (including Wass, MM, CM, MM-COV, CM-COV) will not change.
   ```