Replication Package for

"Learning to Simulate: Generative Metamodeling via Quantile Regression"

by

L.Jeff Hong, Yanxi Hou, Qingkai Zhang, and Xiaowei Zhang

1. General Introduction

This replication package contains the source code used in the numerical experiments of our paper. All experiments were conducted on a Lenovo laptop (Windows 11 Home, Build 26100) equipped with an Intel 12th Gen CPU (14 cores, 20 threads, 2.7GHz), 16 GB RAM, and an NVIDIA GeForce RTX 3060 Laptop GPU (6 GB). The main implementation is written in Python 3.9 and executed through Jupyter Notebooks. In addition to Python, R (version 4.4.1) and MATLAB (R2022a) are also used for specific components of the experiments, including quantile regression estimation and simulation procedures. We note that the numerical results are not sensitive to the exact versions of R or MATLAB, and equivalent outcomes can be reproduced using other recent releases. The package is organized to ensure transparency and reproducibility. For any questions, please contact: `22110690021@m.fudan.edu.cn`.

2. Environment and Dependencies

We recommend setting up the computational environment via `Anaconda`. All Python experiments were developed and executed on Windows using `Python 3.9.12`. The full Python environment specification is provided in `environment_QRGMM_history.yml`, and can be created via:

```
conda env create -n QRGMM -f environment_QRGMM_history.yml.
```

The main Python dependencies used in our experiments (with the versions recorded in the environment file) include:

- `numpy` (1.24.4), `scipy` (1.12.0): numerical computation
- `pandas` (1.4.2): data processing and I/O

- `matplotlib` (3.5.1), `seaborn` (0.11.2): visualization
- `scikit-learn` (1.0.2): linear regression and utility functions
- `statsmodels` (0.13.2): statistical utilities and quantile regression
- `torch` (1.13.1+cu117): deep-learning backend and GPU acceleration (CUDA 11.7), used for CWGAN-GP, Diffusion and Rectified Flow implementations
- `joblib` (1.1.0): parallel computation
- Built-in modules such as `os`, `time`, `random`, and `warnings`: file-system utilities, runtime measurement, random seeding, and warning control

The R scripts rely on the following packages:

- `tictoc`: timing utilities
- `cqrReg`: quantile regression solvers
- `future.apply`: parallel execution for speeding up quantile fits
- `simmer`: discrete-event simulation utilities
- `qrnn`: quantile regression neural network implementation

3. Experiment Settings and Code Structure

The replication package is organized into three main experimental pipelines: Artificial Test Problems, Simulation for Esophageal Cancer Treatments and Simulation for a Bank Queueing System. Each pipeline consists of scripts for data generation, model training, and Jupyter notebooks for performance evaluation. The recommended execution order is as follows:

1. **Artificial Test Problems (`Artificial_Test_Problems`)**

   1.1. `Test_Problem_1_Performance`

   - `TestProblem1_GMMs.py`: Generates training data for Test Problem 1, trains QRGMM, CWGAN, Diffusion, and Rectified Flow models, and generates test observations for evaluation using these models.
   - `Table1and3_TestProblem1.ipynb`: Computes performance metrics based on the generated test samples and reproduces Table 1 and 3 for Test Problem 1 in the main paper.
   - `Figure3_TestProblem1.ipynb`: Visualizes the conditional output distributions and reproduces Figure 3 for Test Problem 1.

- `Figure4_TestProblem1.ipynb`: Evaluates distributional accuracy and reproduces Figure 4 for Test Problem 1.
- Utility functions: `net_utils.py` and `utils.py` (Diffusion and Rectified Flow models), and `wgan.py` (CWGAN model). And they are reused across experiments

1.2. `Test_Problem_2_Performance`

- `TestProblem2_TrainDataGeneration.py`: Generates training datasets for Test Problem 2.
- `TestProblem2_QuantileRegression.R`: Performs quantile regression with basis functions using the `QR.admm` routine from the `cqrReg` R package, and outputs estimated quantile regression coefficients required by QRGMM.
- `TestProblem2_GMMs.py`: Trains QRGMM, CWGAN, Diffusion, and Rectified Flow models for Test Problem 2 using the generated training data and estimated quantile regression coefficients, and generates test observations.
- `Table1and3_TestProblem2.ipynb`: Computes performance statistics and reproduces Table 1 and 3 for Test Problem 2.
- `Figure3_TestProblem2.ipynb`: Visualizes conditional distributions and reproduces Figure 3 for Test Problem 2.
- `Figure4_TestProblem2.ipynb`: Evaluates distributional accuracy and reproduces Figure 4 for Test Problem 2.

1.3. `Convergence_in_Distribution`

- `KS_QRGMM_TestProblem_1.py`: For Test Problem 1, evaluates the convergence in distribution of QRGMM by computing the Kolmogorov–Smirnov (KS) statistic under increasing training sample sizes.
- `KS_QRGMM_TestProblem_2_TrainDataGeneration.py`: Generates training datasets for Test Problem 2 specifically designed for convergence-in-distribution analysis.
- `KS_QRGMM_TestProblem_2_QuantileRegression.R`: Estimates quantile regression coefficients for Test Problem 2 to support QRGMM training in the convergence experiments.
- `KS_QRGMM_TestProblem_2_KSCalculation.py`: Uses trained QRGMM models to generate samples and computes the corresponding KS statistics for Test Problem 2.

- `Figure5_Convergence.ipynb`: Aggregates KS statistics across sample sizes and reproduces Figure 5 in the main paper.

1.4. `Choice_of_m`

- `Figure6_m_effect_on_KS.ipynb`: Analyzes the effect of the number of quantile grid points m and reproduces Figure 6 in the main paper.
- `Quantile_Crossing`
  - `QRGMM_R.py`: Generates samples using both the original QRGMM and the rearranged version QRGMM-R.
  - `QRGMM_R_Performance_Comparisons.ipynb`: Compares the performance of QRGMM and QRGMM-R and reproduces Figure EC.1 and Table EC.1 in the e-companion.
  - `Figure2_QC_Frequency.ipynb`: Evaluates the frequency of quantile crossing and reproduces Figure 2 in the main paper.

2. **Simulation for Esophageal Cancer Treatments** (`Esophageal_Cancer_Simulator`)

2.1. `Treatment_Effects`

- `DataGeneration_Simulator.m`: Generates training and testing data by simulating patient trajectories and treatment outcomes using the esophageal cancer simulator `EsophagealCancerSim.m`.
- `QRGMM_QuantileRegression.R`: Performs quantile regression with basis functions using the `QR.admm` routine from the `cqrReg` R package, and outputs estimated quantile regression coefficients required for QRGMM training.
- `DataGeneration_QRGMM.m`: Uses the trained QRGMM to generate test observations for treatment effect evaluation.
- `DataGeneration_GMMs.py`: Trains CWGAN, Diffusion, and Rectified Flow models, and generates corresponding test observations.
- `DataGeneration_LR.py`: Trains a linear regression benchmark model and outputs its predicted treatment outcomes on the test set.

- `Table2_TreatmentEffects.ipynb`: Computes conditional average treatment effects, performs treatment ranking, and reproduces Table 2 in the main paper.
- `Table3andEC3_ECsimulator.ipynb`: Computes additional performance statistics and reproduces Tables 3 and EC.3.
- `FigureEC4_ECsimulator.ipynb`: Evaluates distributional accuracy of generated outcomes and reproduces Figure EC.4 in the e-companion.
- Utility functions: `Poly.m` (polynomial basis functions), `QRGMM_fun.m` (unconditional QRGMM sampling), `QRGMM_xstar_fun.m` (conditional QRGMM sampling), `net_utils.py` and `utils.py` (Diffusion and Rectified Flow models), and `wgan.py` (CWGAN model).

2.2. `Probability_of_Correct_Selection`

- `BruteForceSim.m`: Computes ground-truth treatment performance by exhaustive simulation, and saves the true QALY values in `QALY_true.mat`.
- `PersonalizedTreatment.m` (lines 1–257): Initializes the experimental environment and computes parameters required for TS ranking-and-selection procedures using `find_h_het_PCSE_EC.m`, `select_het_EC.m`, and `kSystem_EC.m`. Implements the TS procedure and saves intermediate results in `data_het.mat`. Also implements the KN+Simulator procedure via `KN_simulator.m`, and generates training data for generative metamodels.
- `RS_QuantileRegression.R`: Performs quantile regression with basis functions using the `QR.admm` routine from the `cqrReg` R package, and outputs estimated quantile regression coefficients required by QRGMM for ranking-and-selection tasks.
- `RS_CWGAN.py`: Trains the CWGAN model and applies it to the ranking-and-selection task.
- `RS_DiffRect.py`: Trains Diffusion and Rectified Flow models and applies them to the ranking-and-selection task.
- `PersonalizedTreatment.m` (lines 258–end): Uses utility functions `RS_QRGMM.m`, `QRGMM_fun_Y1.m`, and `QRGMM_fun_Y2.m` to perform ranking and selection using QRGMM, and computes the corresponding results for CWGAN, Diffusion, and Recti-

fied Flow models.

- `Table4_RS_results_summary.ipynb`: Summarizes results and and reproduces Table 4 in the main paper.
- Utility functions: `EsophagealCancerSim.m`, `Poly.m`, `net_utils.py`, `utils.py`, and `wgan.py`. Serve the same purposes as described above and are reused across treatment effect evaluation and ranking-and-selection experiments.

3. **Simulation for a Bank Queueing System (`Bank_Simulator`)**

- `BankSimulator.R`: Implements the bank queueing system simulator and generates training and test datasets for all competing models.
- `BankSimulator_QRNNGMM.R`: Implements a neural-network-based QRGMM using the `qrnn` R package, and generates unconditional test observations for performance evaluation.
- `BankSimulator_QRNNGMM_OnlineGenerationTiming.R`: Implements the neural-network-based QRGMM using the `qrnn` R package, generates conditional test observations, and records sample generation time for efficiency comparison.
- `BankSimulator_CWGAN.py`: Trains the CWGAN model on the simulated training data, generates conditional test samples, and records online generation time.
- `BankSimulator_DiffRect.py`: Trains Diffusion and Rectified Flow models on the simulated training data, generates test observations, and records online generation time.
- `TableEC2_and_FigureEC2.ipynb`: Computes generation time summaries and reproduces Table EC.2 and Figure EC.2 in the e-companion, comparing QRGMM, CWGAN, Diffusion, and Rectified Flow.
- `FigureEC3.ipynb`: Evaluates distributional accuracy of generated outcomes and reproduces Figure EC.3 in the e-companion.
- Utility functions: `net_utils.py` and `utils.py` (Diffusion and Rectified Flow models), and `wgan.py` (CWGAN model).

4. Special Notes

- **Model hyperparameter references and training epochs.** The hyperparameter settings of the CWGAN model follow the recommendations in Athey et al. (2024), while the hyperparameters for the Diffusion and Rectified Flow models are adopted from Liang and Chen (2024). Unless otherwise specified, all these models are trained for 1,000 epochs, with a typical training sample size of 10,000. An exception is the `Probability_of_Correct_Selection` experiments, where all generative models are trained for 500 epochs using a larger training dataset of 100,000 samples, to balance computational cost and statistical accuracy.
- **Parallel computation and acceleration.** The quantile regression fitting step in QRGMM is naturally parallelizable across quantile levels. In Python, parallelization is implemented using the `joblib` package, while in R it is implemented using `future.apply`. For CWGAN, Diffusion, and Rectified Flow models, GPU-based parallel computation with CUDA is used to accelerate training and online sample generation.
- **Reproducibility:** To facilitate reproducibility, random seeds are fixed across all runs.

References

Athey, S., G. W. Imbens, J. Metzger, and E. Munro (2024). Using Wasserstein generative adversarial networks for the design of Monte Carlo simulations. *Journal of Econometrics 240*(2), 105076.

Liang, E. and M. Chen (2024). Generative learning for solving non-convex problem with multi-valued input-solution mapping. *The Twelfth International Conference on Learning Representations*.