PROJECT REPORT

1. The correctness of the simulation code

Code:main.py

(1a) Test Result

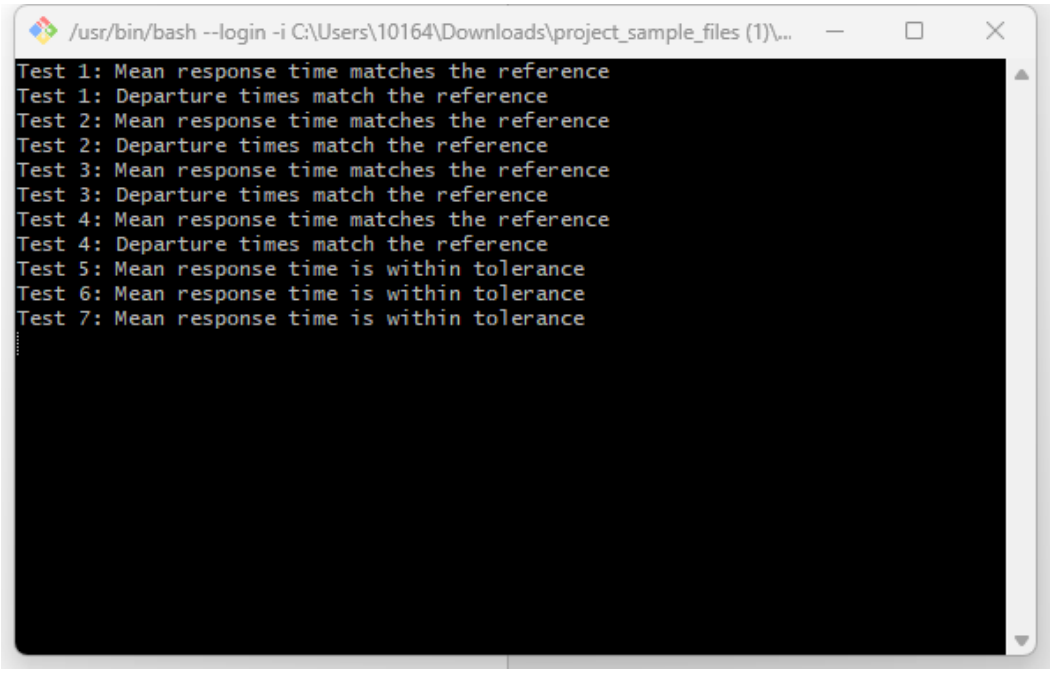
I temporarily modified the file run_test.sh to conduct batch testing and comparison for my model. All seven tests were successfully passed.

There are the run_test.sh and the test results:

run_test.sh

```
python main.py 1
python main.py 2
python main.py 3
python main.py 4
python main.py 5
python main.py 6
python main.py 7
python cf_output_with_ref.py 1
python cf_output_with_ref.py 2
python cf_output_with_ref.py 3
python cf_output_with_ref.py 4
python cf_output_with_ref.py 5
python cf_output_with_ref.py 6
python cf_output_with_ref.py 7

read -n 1 -s
```



All the tests provided are successfully passed.

(1b) Verification of Random Number Generators

Code: verify_distributions.py

For this design problem, you will assume the following parameter values:

- Number of servers: $n = 8$
- For inter-arrival times: $\lambda = 1.6$, $a_{2\ell} = 0.9$, $a_{2u} = 1.1$
- For the number of sub-jobs per job: the sequence p_1, p_2, p_3, p_4,p_5, p_6 is 0.22, 0.28, 0.3, 0.08, 0.07, 0.05.
- For the service time per job: $\mu = 0.8$, $\alpha = 0.8$.

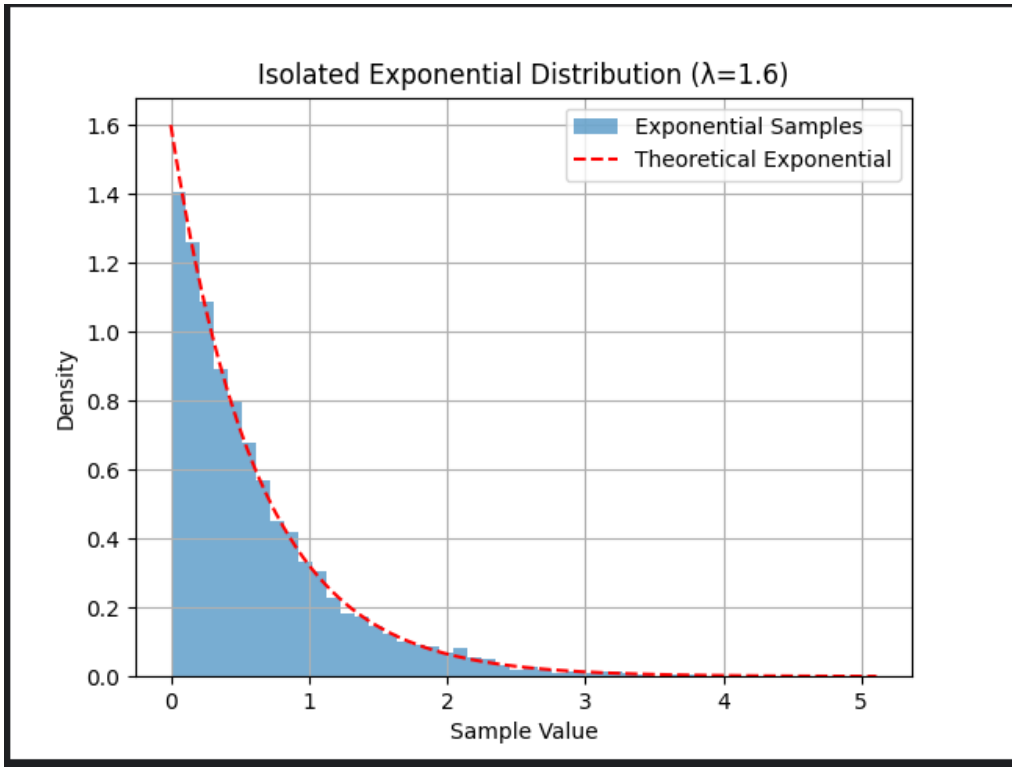
For the random mode, conduct the test using the provided parameters.

Separate Verification:

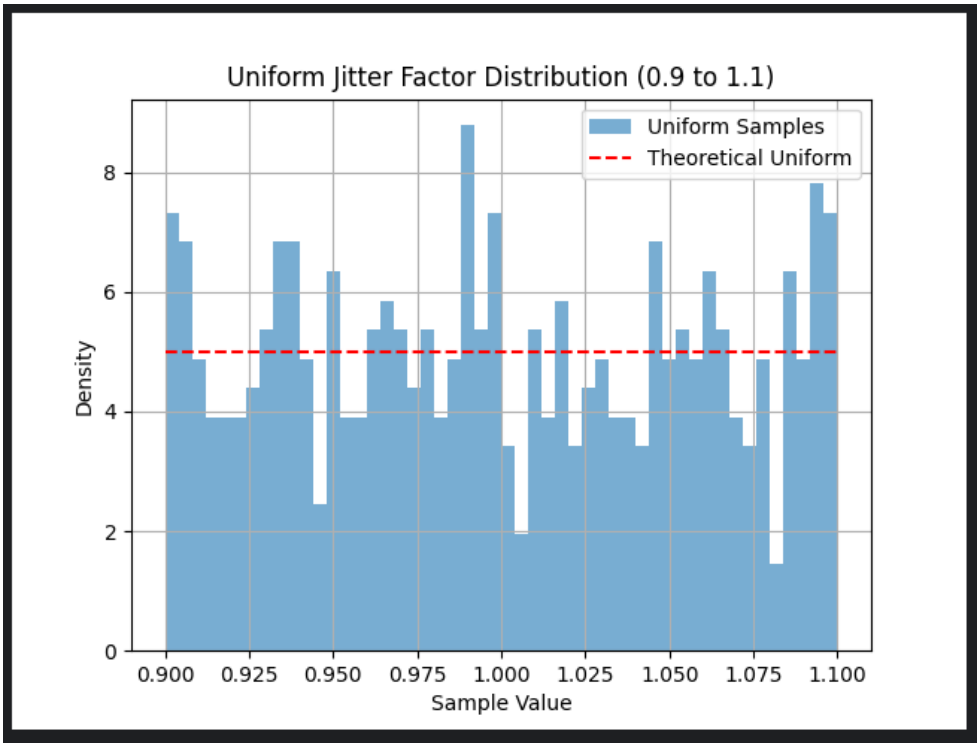
To ensure that the simulation inputs in random mode are statistically valid, we verified the correctness of the random number generators used to produce task interarrival times, sub-job counts, and sub-job service times. We conducted both visual inspections and statistical tests to validate the distributions.

Task interarrival times are produced by multiplying an exponential distribution with rate parameter λ = 1.6 by a uniform jitter factor from the range [0.9, 1.1]. To validate this, we separately evaluated the exponential and uniform components.

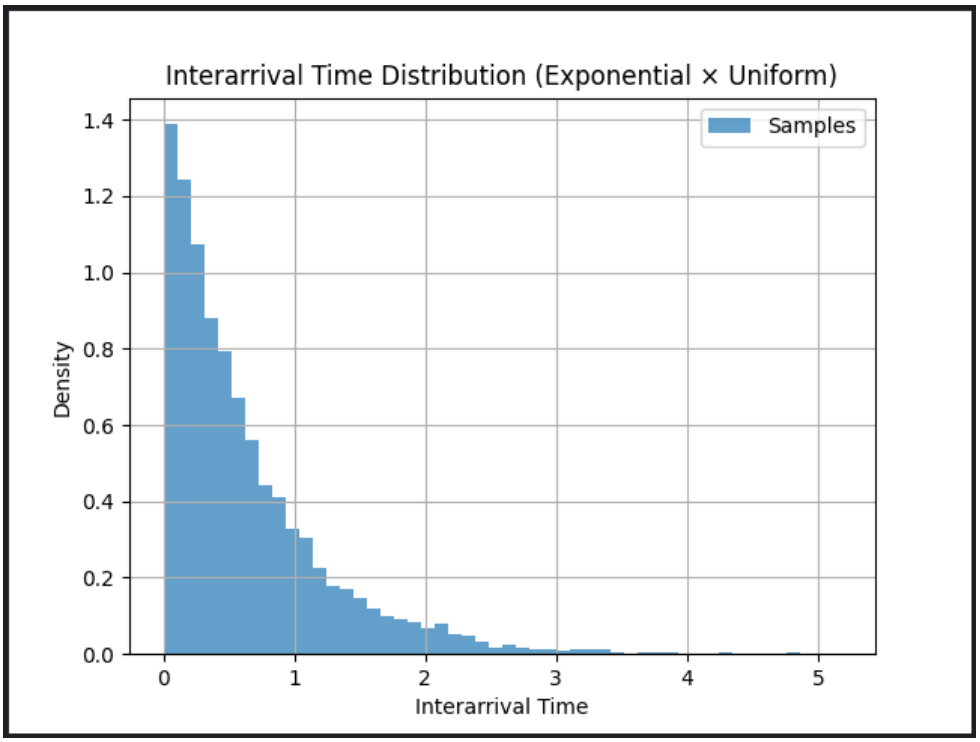
The **exponential samples**, after normalization, closely follow the theoretical exponential distribution:



- The **uniform jitter factors** follow the Uniform(0.9, 1.1) distribution as expected:



- The final interarrival times (product of both) are also visualized:



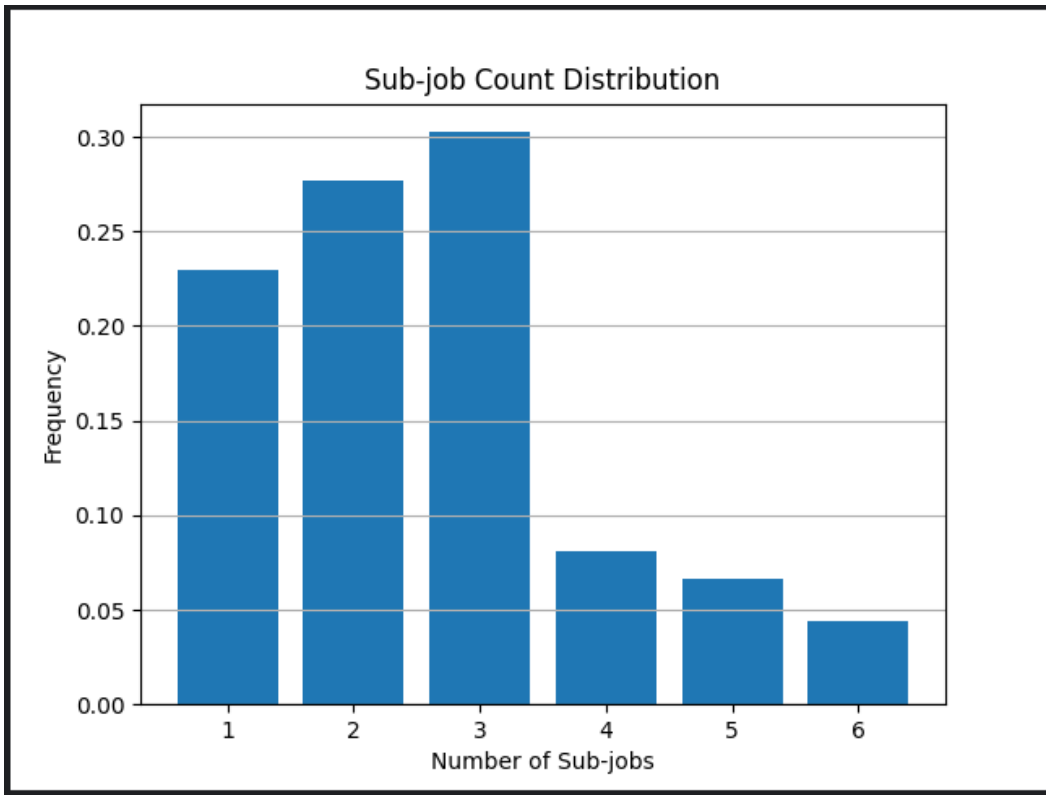
- KS test on interarrival time samples:
  P-value = 0.2911
  This value > 0.05, indicating that the samples conform to the expected distribution.

Sub-job Count Distribution

The number of sub-jobs per task is randomly selected from a discrete distribution over {1, 2, ..., 6} using the given probabilities:

[0.22, 0.28, 0.3, 0.08, 0.07, 0.05]

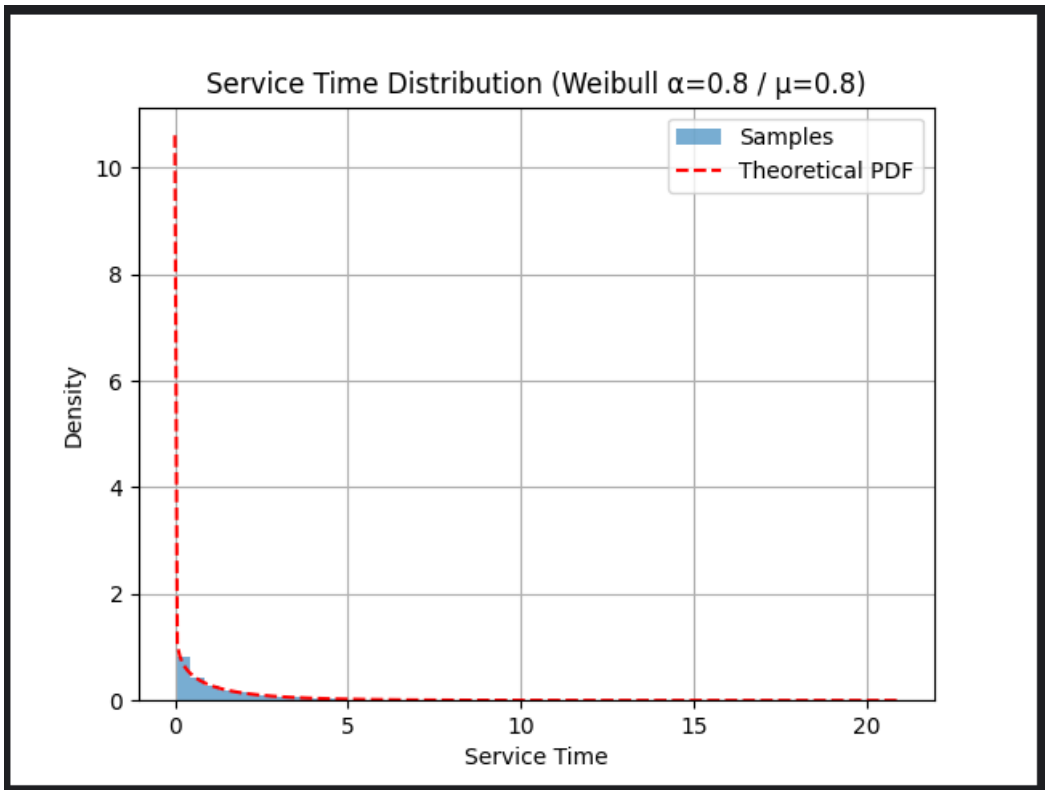
A histogram was generated to compare the observed frequencies with the expected values:



Service Time Distribution

Each sub-job's service time is sampled from a Weibull distribution with shape parameter α = 0.8 and scaled by μ = 0.8.

The sample distribution aligns well with the theoretical Weibull PDF:



- KS test on service time samples:
  P-value = 0.5826
  The result supports the hypothesis that samples follow the target Weibull distribution.

```
[KS Test Results]
P-value for exponential interarrival samples = 0.2911
P-value for subjob count distribution = N/A (discrete distribution, KS not applicable)
P-value for Weibull service time samples = 0.5826
```

2. Verification of the reproducibility of simulation results

To ensure the reproducibility of the simulation results, we set a random number seed (seed) in the program and used the default generator of NumPy, np.random.default_rng(25). This ensures that, under the same input parameters, the random number sequence generated each time is exactly the same, thereby obtaining consistent simulation results.

In the generate_random_trace() function:

```python
def generate_random_trace(time_end, lambda_rate, a2l, a2u, subjob_probs, mu, alpha, seed=None):
    arrivals = []
    services = []
    time = 0
    rng = np.random.default_rng(seed)  # use fixed seed if provided

    while time < time_end:
        interarrival = rng.exponential(1 / lambda_rate) * rng.uniform(a2l, a2u)
        time += interarrival
        if time >= time_end:
            break

        arrivals.append(time)
        num_subjobs = rng.choice(np.arange(1, len(subjob_probs)+1), p=subjob_probs)
        sub_services = rng.weibull(alpha, num_subjobs) / mu
        services.append(sub_services)

    return np.array(arrivals), services
```

All samples (arrival intervals, number of sub-tasks, service times) are generated based on a fixed seed.
The seed value we have chosen is 25, which can be replicated by others.
Whether it is verifying the charts or the final MRT indicators, as long as the parameters remain unchanged, the results can be consistently reproduced.
This method ensures that the charts and performance evaluations we submit are verifiable and consistent.

3.Threshold Selection for Minimizing Mean Response Time

Code: design_experiment.py

(3a) Simulation and Data Processing Parameters

In MRT evaluation across different h values, we did not fix the random seed to avoid biasing results by using identical random sequences. This helps ensure the robustness and fairness of the comparison.

In order to determine the optimal threshold h, we conducted multiple sets of simulation experiments. The following are the parameter settings and their rationality:
Set the simulation duration to 4000 units of time to ensure that the system reaches a steady state.
The simulation time of 4000 units ensures that the system moves beyond the initial transient phase and reaches steady-state behavior. This length was chosen based on empirical observations that MRT stabilizes after the initial period.
Each h value is simulated 30 times consecutively to reduce randomness.
Each threshold h was tested with 30 independent simulation runs to account for stochastic variability and to allow for the calculation of confidence intervals. The value 30 was chosen as

a balance between statistical robustness and computational efficiency.
Traversal h
h ranges from 0 to 8, corresponding to the number of servers;
The threshold h was varied from 0 to 8, where 0 means only tasks with one sub-job receive priority, and 8 implies no priority differentiation. This range fully captures all meaningful configurations in an 8-server system.
Fixed simulation parameters:
Server count: 8
Arrival rate λ = 1.6; perturbation range [0.9, 1.1]
Sub-task distribution: 6 discrete probabilities
Service time: Weibull (α = 0.8) / μ = 0.8

To ensure statistical reliability, we calculated 95% confidence intervals (CIs) for the mean response time (MRT) of each threshold h. When evaluating the best threshold, we favored those whose MRTs not only had the lowest means but also had **non-overlapping CIs** with higher-MRT candidates.
This indicates that the difference is statistically significant, rather than due to random fluctuations in the simulation.

(3b) Confidence Interval-Based Selection of Threshold h
To determine the most appropriate threshold h for minimizing the mean response time (MRT), we conducted simulations for each h ∈ {0, 1, ..., 8}, with 30 repetitions per setting. For each h, we computed
the sample mean MRT and its 95% confidence interval using the standard error of the mean:

$$CI = [\mu - 1.96 \times (s / \sqrt{n}), \mu + 1.96 \times (s / \sqrt{n})]$$

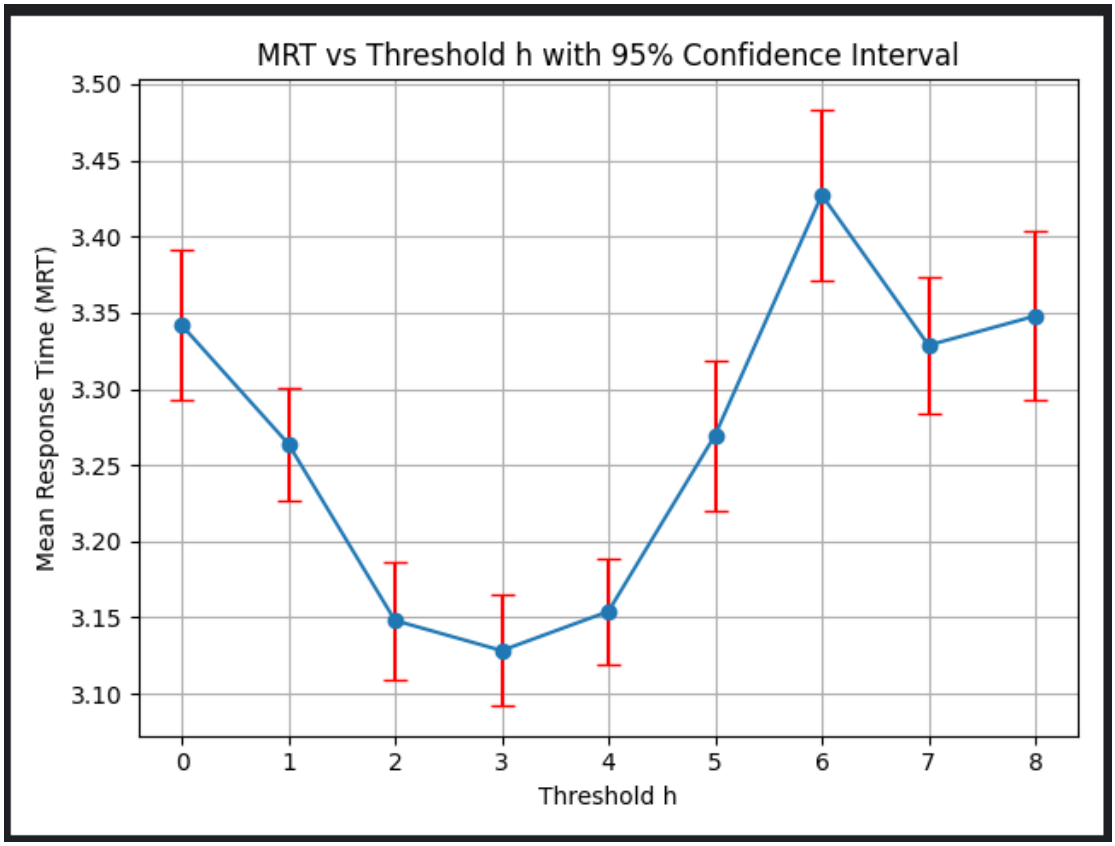
Where:
- μ is the sample mean MRT,
- s is the sample standard deviation,
- n = 30 is the number of repetitions.

We plotted the MRT against h along with their confidence intervals (see figure below).
The optimal value of h is selected not only based on the lowest mean MRT, but also by considering whether its confidence interval does not overlap with those of neighboring h values. This statistical significance gives us more confidence that the improvement in performance is not due to random fluctuations.

In summary, the chosen h value minimizes MRT while showing statistically robust separation from other alternatives.

```
Running simulations to find best h...

h = 0: MRT = 3.3419, 95% CI = [3.2927, 3.3911]
h = 1: MRT = 3.2640, 95% CI = [3.2269, 3.3011]
h = 2: MRT = 3.1481, 95% CI = [3.1093, 3.1869]
h = 3: MRT = 3.1283, 95% CI = [3.0920, 3.1646]
h = 4: MRT = 3.1540, 95% CI = [3.1195, 3.1885]
h = 5: MRT = 3.2697, 95% CI = [3.2203, 3.3190]
h = 6: MRT = 3.4273, 95% CI = [3.3712, 3.4833]
h = 7: MRT = 3.3287, 95% CI = [3.2844, 3.3729]
h = 8: MRT = 3.3478, 95% CI = [3.2925, 3.4032]
```



(This is one of the test results.)

However, I conducted ten tests to find the optimal h value. Among them, the best h value was 3 in 9 tests and 4 in 1 test. Therefore, the final test h value obtained was 3.