TRANSPORT AND TELECOMMUNICATION INSTITUTE

System Analysis and Modelling

Practical Assignment 1

Final Report

Student: Julija Belova

St. code: 78727

Lecturer: Mihails Savrasovs

RIGA

TABLE OF CONTENTS

I INTRODUCTION

1.Tasks

This practical assignment involves the development and analysis of simulation models for a system described in the provided context. The tasks include designing the system, putting simulation models into practice using a variety of ways, verifying random number generators, producing output data based on certain datasets, and contrasting the outcomes of various simulation techniques.

Making a conceptual model of the system using the given description is the first task. After that, a simulation model must be developed in any programming language using a discrete event simulation technique. To guarantee accuracy and dependability, the random number generators employed in the simulation model must then be put to the test and verified. I used Python.

After the model has been validated, the first simulation model (Model1) based on a given dataset will be used to create output data, especially Measures of Effectiveness (MoE). Two further simulation models must also be created: Model2, which uses the GPSS simulation language, and Model3, which uses the AnyLogic simulation system. Both models use the same dataset as their basis for output data collection.

Following the collection of data from each of the three models, the outcomes must be contrasted and examined. The purpose of this comparison analysis is to provide light on how the system behaves and performs while using various simulation techniques. The analysis's conclusions should then be reviewed, emphasising their significance and ability to guide system optimisation and decision-making.

2. Individual Variant

Unfortunately, some parameters from the option gave an error in the form of a negative timer, so it was decided to slightly change two values, normal(2,2) to normal(2,0.2) and normal(2,1.5) to normal(2,0.5) . This error also occurred with my classmate, who had the same values, but a different option

The individual variant for this practical assignment is number 6. This variant has the following dataset:

Table 1. Individual variant

Dataset	*I1*	*I2*	*P1*	*P2*	*Q*	*MoE1*	*MoE2*
7	Erlang (2,3)	Normal (2,0.2)	Normal (2,1.5)	Normal(1.5, 0.5)	LIFO	Downtime factor	Average of jobs in queue

II RANDOM NUMBER GENERATOR

In order to develop our model we need to implement a random number generators with Erlang and Normal Distribution

1. RNG for exponential distribution:

1.1 Implementation

The Erlang distribution was implemented using a custom random number generator based on the Linear Conruential Generator method.

1.2 Comparison of Empirical and Theoretical Parameters

Table 1. Empirical and Theoretical Parameters of Erlang Distribution

Parameter	***Empirical***	***Theoretical***
MEAN	1.902348	2
STD	3.205693	3

The empirical and theoretical parameters of the Erlang distribution were compared. While the shape parameter (mean) is slightly lower in the empirical distribution compared to the theoretical one, the scale parameter (std) is slightly higher.

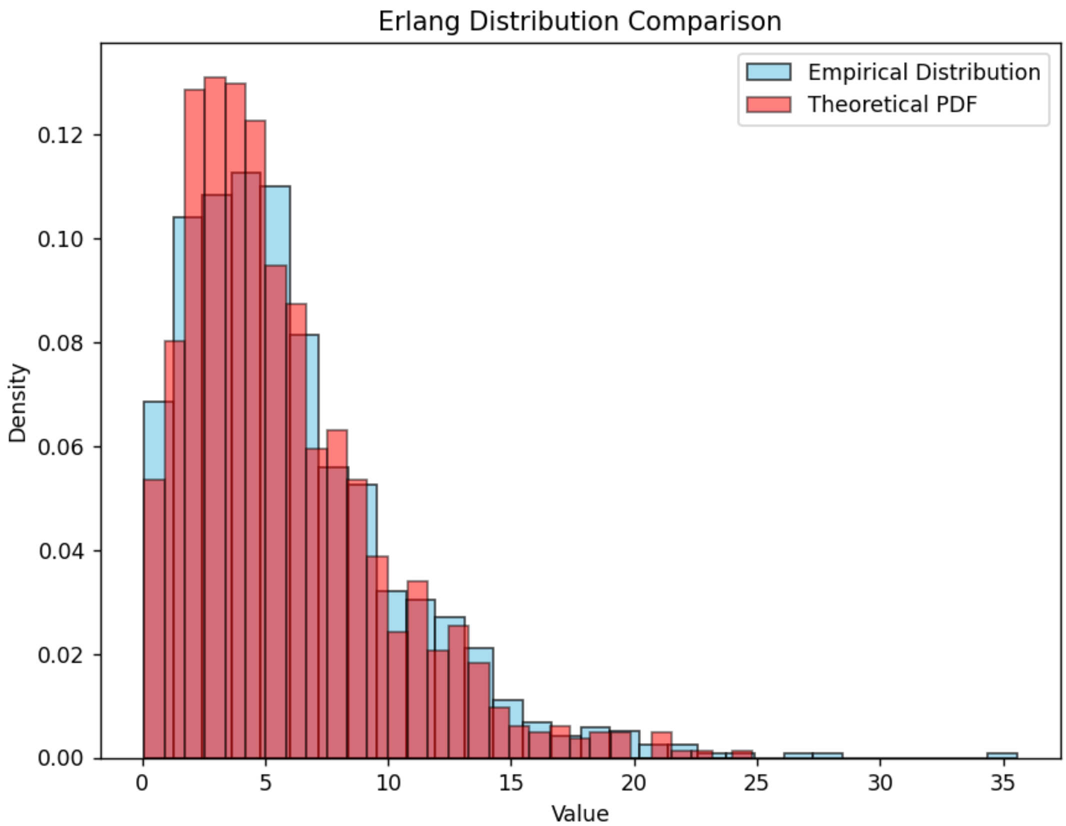

Figure 1. Erlang Distribution Comparison

1.3 Kolmogorov-Smirnov Test

Null Hypothesis: The sample follows the Erlang distribution with specified parameters.

Alternative: Sample distribution is significantly different from the expected Erlang distribution.

Significance Level: 5% or 0.05

Result: D = 0.020, p-value = 0.794

Conclusion: Fail to reject the null hypothesis

```
Kolmogorov-Smirnov test: statistic = 0.02036032406472965, p-value = 0.7935191226207714
Fail to reject the null hypothesis
```

1.4 Chi-Square Test

Null Hypothesis: There is no significant difference between the observed frequencies and the expected frequencies, and the sample follows the Erlang distribution with specified parameters.

Alternative: There is a significant difference between the observed frequencies and the expected frequencies.

Significance Level: 5% or 0.05

Result: X-squared = 1.561, df = 20, p-value = 1.000

Conclusion: Fail to reject the null hypothesis

```
Chi-square test: statistic = 1.560982999216557, p-value = 0.9999999999999604
Fail to reject the null hypothesis
```

1.5 Conclusions

Based on the Kolmogorov-Smirnov and Chi-Square tests, along with the visual inspection of the histogram, we conclude that the implemented Erlang distribution generator is capable of generating a random sequence of values that closely follow the Erlang distribution with the specified parameters.

2. RNG for normal distribution:

2.1 Implementation

The normal distribution was implemented using a custom random number generator based on the Linear Conruential Generator method.

1.2 Comparison of Empirical and Theoretical Parameters

Parameter	***Empirical***	***Theoretical***
MEAN	1.997963	2
STD	0.186853	0.2

Table 3. Empirical and Theoretical Parameters of Normal Distribution

The empirical and theoretical parameters of the Normal distribution were compared. While the mean parameter is very close between empirical and theoretical distributions, the standard deviation is slightly lower in the empirical distribution compared to the theoretical one.

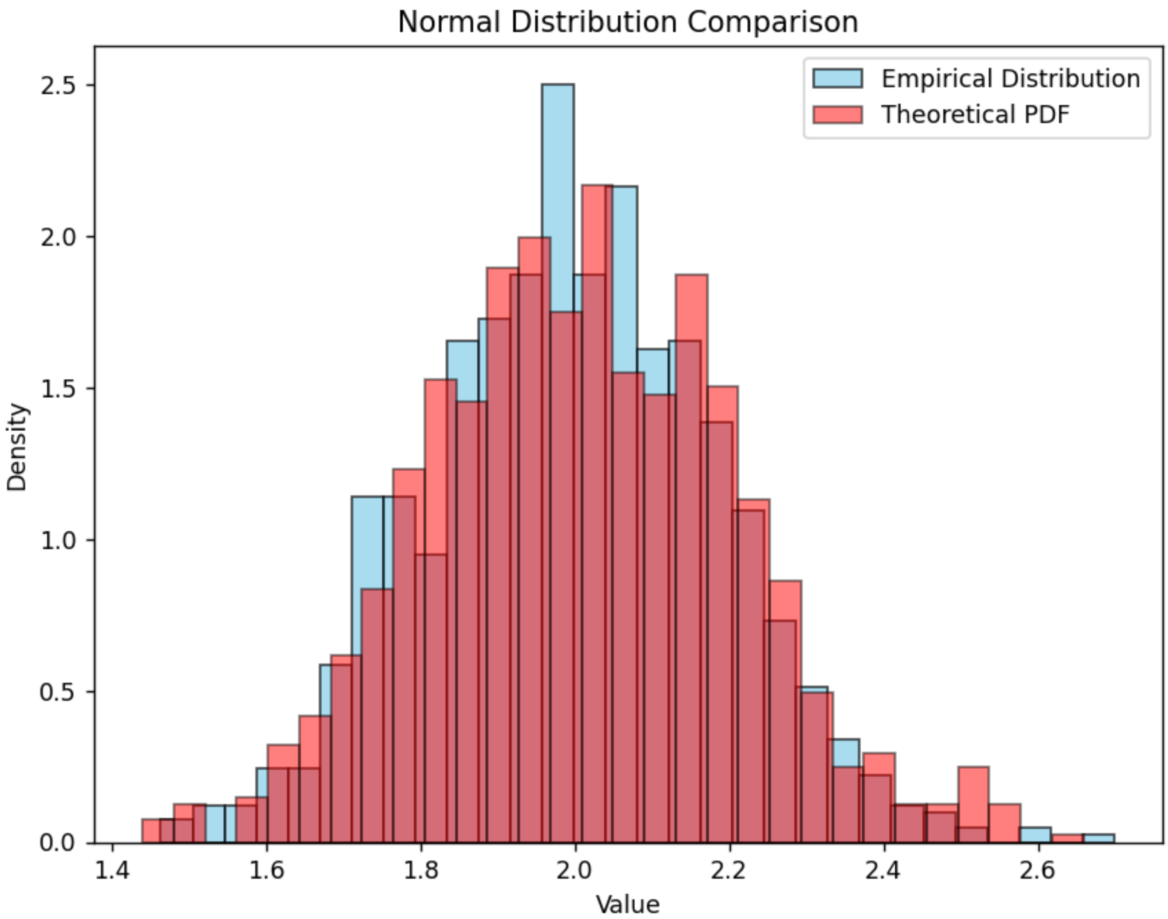

Figure 2. Normal Distribution Comparison

1.3 Kolmogorov-Smirnov Test

Null Hypothesis: The sample follows the Normal distribution with specified parameters.

Alternative: Sample distribution is significantly different from the expected Normal distribution.

Significance Level: 5% or 0.05

Result: D = 0.028, p-value = 0.413

Conclusion: Fail to reject the null hypothesis

```
Kolmogorov-Smirnov test: statistic = 0.027848061922657208, p-value = 0.4125482772058511
Fail to reject the null hypothesis
```

1.4 Chi-Square Test

Null Hypothesis: There is no significant difference between the observed frequencies and the expected frequencies, and the sample follows the Normal distribution with specified parameters.

Alternative: There is a significant difference between the observed frequencies and the expected frequencies.

Significance Level: 5% or 0.05

Result: X-squared = 21.542, df = 20, p-value = 0.839

Conclusion: Fail to reject the null hypothesis

```
Chi-square test: statistic = 21.542035441594102, p-value = 0.8386490326468667
Fail to reject the null hypothesis
```

1.5 Conclusions

Based on the Kolmogorov-Smirnov and Chi-Square tests, along with the visual inspection of the histogram, we conclude that the implemented Normal distribution generator is capable of generating a random sequence of values that closely follow the Normal distribution with the specified parameters.

III CONCEPTUAL MODEL

1. Description Of The System And Conceptual Model

Discrete event simulation was used to analyse a server system that could do two different kinds of tasks. Type I1 tasks arrive to the server at an exponential rate defined by an Erlang distribution with shape and scale parameters of 2 and 3, respectively; type I2 jobs come at an exponential rate defined by a normal distribution with mean 2 and standard deviation 0.2.Tasks with different processing durations are handled by the server's processor: P1 and P2 tasks have processing times that are distributed normally, with mean and standard deviation values of (2, 1.5) and (1.5, 0.5), respectively. Last-in, first-out (LIFO) is how the queue is structured. Two metrics—the average number of tasks in the queue (MOE2) and the downtime factor (MOE1)—are used to assess how successful the system is.

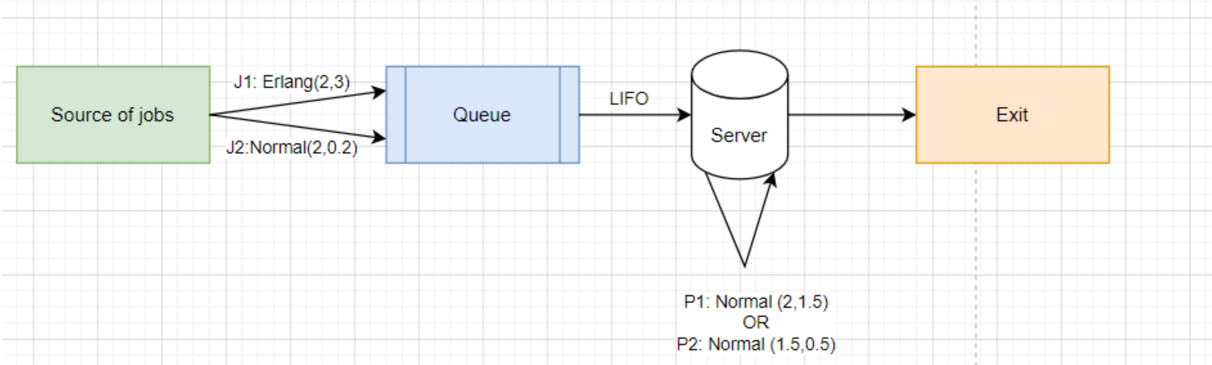

Figure 3. Conceptual diagram of the system

2. Model Development Using The Programming Language

2.1 Flowchart algorithm of the model

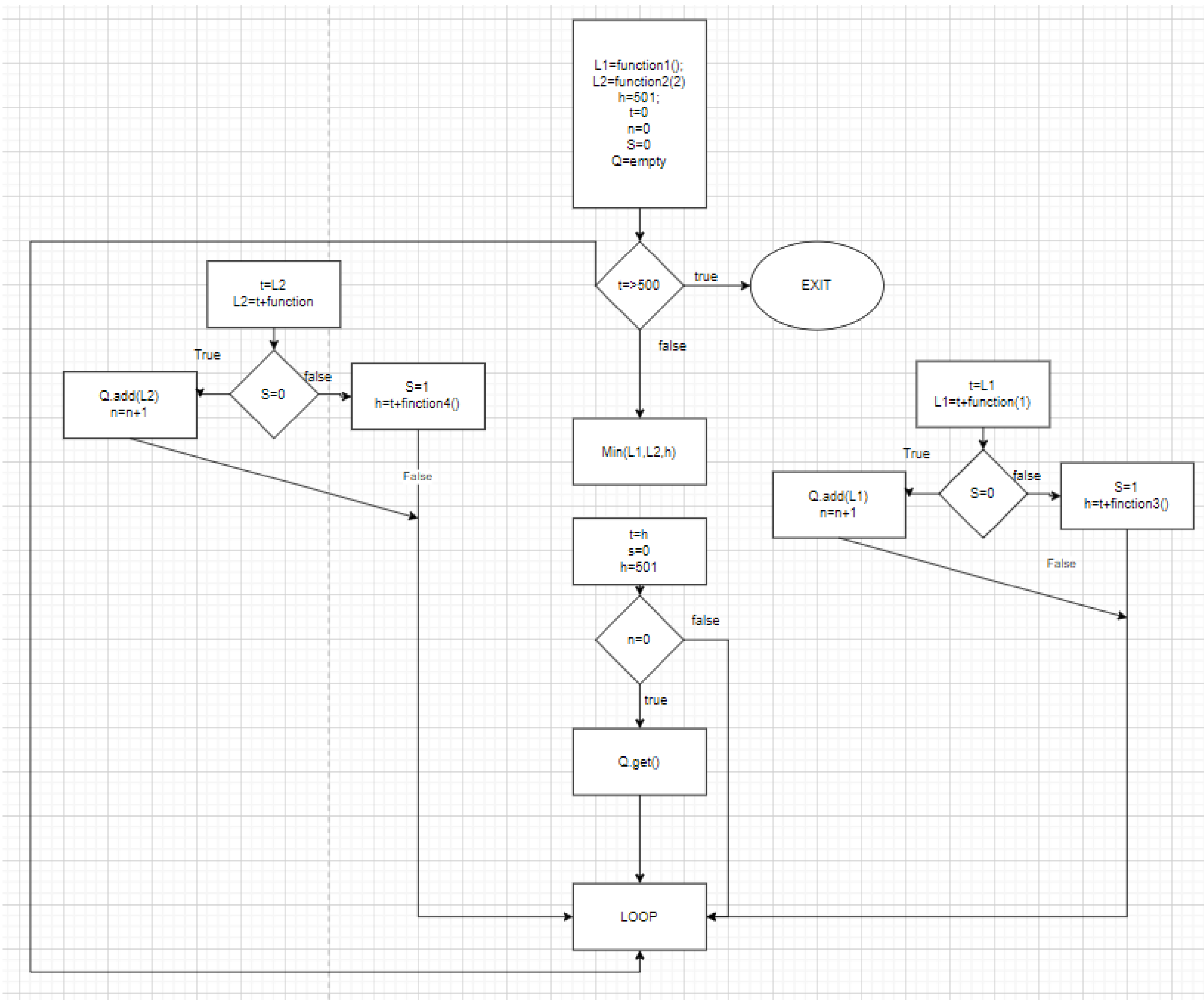

Figure 4. Flowchart model1

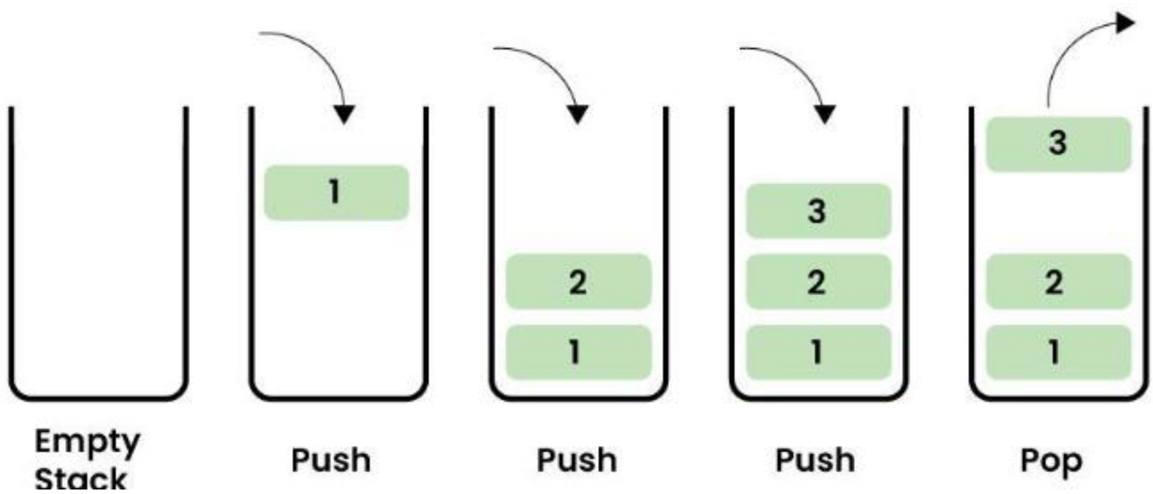

Figure 5. LIFO priciple

IV MODEL 1 DEVELOPMENT

Model is written in Python 3.12.2. The source code for the program is available in the Model1.py file located in the Model1 folder.

1. Description

Throughout the 500 time units (minutes) while the simulation runs, a variety of events are captured and saved in a data structure known as simulation_data. This data is converted into a structured format for analysis at the end of the simulation.

The random number generator seed, simulation period, and accuracy settings are all established at the beginning of the experiment. The simulation results are guaranteed to be consistent and repeatable by these settings.

The normal() and erlang() methods, respectively, are used to generate random integers from the normal and exponential distributions. These routines are designed to mimic the server's processing timings as well as the arrival times of the J1 and J2 jobs.

The simulation follows a last-in, first-out (LIFO) discipline and uses a queue data structure to handle outstanding jobs. Accordingly, the most recent job submitted to the queue is serviced first, and tasks are handled in the order they are added to it.

The core simulation process is powered by the simulation loop, which is contained within a while loop. Until the specified simulation end time is achieved, it repeats the occurrences.

New jobs are produced based on the arrival distributions of each iteration of the loop. The next event time is set by taking the minimum of the J1 and J2 task arrival timings and the server task completion time that is currently running.

The simulation modifies pertinent variables and queues based on the kind of event (J1 arrival, J2 arrival, or completion of server task).

The simulation monitors a number of variables, such as the system time, the length of the task queue, and the availability or busyness of the server. Throughout the simulation, these parameters are crucial in capturing the dynamics of the system.

Measures of effectiveness (MOE1 and MOE2) are calculated at the conclusion of the simulation using the data that has been gathered. Metrics like average queue length and server downtime offer insights into system performance.

All things considered, the simulation model offers an extensive framework for researching and examining how the task processing system behaves in different scenarios.

2. Simulation results

```
Simulation process table template
Num Time Event J1 J2 St S
1  0.49830417846969316 J1 0.0 1.7765836924837444 Busy 0
2  1.4328857071095185 E1 0.8353801095421887 1.3947067019996129 Free 0
3  2.618533037282332 E2 0.7085541425572162 2.599393345383053 Free 0
4  3.9837119298338073 E2 1.87313357569071 3.1786123376207907 Free 0
5  4.708693672368348 E2 3.4360617467384804 5.256327753017666 Free 0
6  6.1228363541671165 E2 4.361401633867832 5.437316818174656 Free 0
7  8.031605414030087 E2 5.6990158123056815 6.299124991746158 Free 0
8  8.9819809598007 E2 7.825773183046666 9.123540100939916 Free 0
9  10.260497423502366 E2 9.168608520398836 9.7384660383374 Free 0
10 11.566024413758559 E2 9.050415361474554 11.341306375192248 Free 0
...
345 488.66423957679547 E2 488.42235256927944 489.1359248057712 Free 0
346 489.6990229665152 E2 488.4438212997798 489.82124370377346 Free 0
347 491.1412933842264 E2 488.3980726236189 489.9454994525625 Free 0
348 492.06933221280985 E2 491.42047390719404 492.1488019964479 Free 0
349 493.42120396810253 E2 491.1948603474791 492.77988951130857 Free 0
350 495.28732881284265 E1 493.0950624437138 493.4921430648131 Free 0
351 496.09700528047426 E2 495.1084662176717 496.3483677335092 Free 0
352 498.1371360677768 E1 494.29875119472104 496.09997423247637 Free 0
353 499.477454284039 E2 497.38943836202947 498.5638994865045 Free 0
354 500.9164153716674 E2 inf inf Free 0

MOE1 Downtime factor: 0.706
MOE2 Average of jobs in queue: 0.0
```

Figure 6. Console output example

Comments about the table:

Num - Each line of input number

• Time – current time of the simulation

• Event – indicates the name of an event: Start – simulation start; E1 – arrival of Job 1; E2 – arrival of Job 2;

• J1 – the arrival time of the next job of type 1

• J 2 – the arrival time of the next job of type 2

Stop time - the time, then processing of the job will be completed. Initially set to <501

• St – status of the server (free or busy)

• S – the queue length

• MoE1 – Downtime factor

• MoE2 – Average of all jobs in queue

3. MoE statistics

Table 2. Model 1 MoE statistics

Random seed	*Moe 1 - Downtime factor*	*MoE 2 – Average of all jobs in queue*
1	0.69	0
2	0.678	0
3	0.706	0
4	0.68	0
5	0.694	0
6	0.678	0
7	0.686	0
8	0.69	0
9	0.684	0
10	0.698	0
Statistics	Average: 0.689 Mode: 0.69 Median: 0.688 Standard deviation: 0.001 95% interval: 0.678 to 0.704	Average: 0 Mode: 0 Median: 0 Standard deviation: 0 95% interval: 0 to 0

V MODEL 2 DEVELOPMENT

Model is written in GPSS World Student Version. The source code for the program is available in the Model 2.gps file located in the Model 2 folder.

1. Code of the model

The GPSS code that is given models the flow of jobs through a processing server by simulating a queuing system. This system has two different job categories, j1 and j2, each with its own arrival patterns and features.

J1 tasks are first generated by the simulation using a gamma distribution with parameters (1, 0, 1/3, 2). The inter-arrival periods of j1 jobs are controlled by this distribution, which also controls how often they enter the system. In contrast, j2 tasks are produced with their inter-arrival intervals determined by a normal distribution with parameters (1, 2, 0.2).

Jobs join a queue called "line" as soon as they are added to the system and wait for the server to process them. Last-in-first-out (lifo) queuing discipline is used to make sure that the job that came most recently is given priority for processing when the server becomes available.

The server takes the next job out of the queue and starts processing it as soon as it is free. The server allots the allotted processing time to the task, modelling the amount of time needed to finish it. When the work is finished, the server lets it back into the system so it may finish the simulation.

A timer that generates pseudo-transactions every 500 time units drives the simulation forward within a predetermined time limit. By using a temporal control mechanism, the simulation will end when it has simulated a long enough period of system activity.

In conclusion, the GPSS code encompasses a whole simulation of a queuing system, including job creation, processing, and system dynamics. This full simulation offers important insights into the functionality and behaviour of the modelled system.

```
; --- Job Generation ---
; Generate entities representing type J1.
GENERATE      (Gamma(1,0,1/3,2))          ; Determine next arrival time for J1.
ASSIGN        ptime,(Normal(1,2,0.15))    ; Define processing time for J1.
TRANSFER      ,WAIT                       ; Proceed to the WAIT block.

; Generate entities representing type J2.
GENERATE      (Normal(1,2,0.2))   ; Determine next arrival time for J2.
ASSIGN        ptime,(Normal(1,1.5,0.5))  ; Define processing time for J2.

; --- Job Processing ---
WAIT   LINK   LINE,LIFO,PROC          ; Join the queue while the server is busy.
PROC   SEIZE  SRV      ; Acquire the server for processing.
ADVANCE       P$ptime ; Simulate the job processing time.
RELEASE       SRV      ; Release the server upon job completion.
UNLINK        LINE,PROC,1     ; Move to processing the next queued job.
TERMINATE 0   ; Conclude this job entity.

; --- Simulation Termination ---
; Terminate the simulation after 500 time units (minutes).
GENERATE      500
TERMINATE     1
```

2. Code of the model

The simulation ran successfully every time, producing appropriate results.

```
                GPSS World Simulation Report - Practical Assignment 1.89.1


                     Wednesday, April 24, 2024 14:26:44

           START TIME           END TIME  BLOCKS  FACILITIES  STORAGES
                0.000            500.000    13         1          0


              NAME                       VALUE
           LINE                        10001.000
           PROC                            7.000
           PTIME                       10000.000
           SRV                         10002.000
           WAIT                            6.000


 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                    1    GENERATE           770             0       0
                    2    ASSIGN             770             0       0
                    3    TRANSFER           770             0       0
                    4    GENERATE           251             0       0
                    5    ASSIGN             251             0       0
WAIT                6    LINK              1021           761       0
PROC                7    SEIZE              260             0       0
                    8    ADVANCE            260             1       0
                    9    RELEASE            259             0       0
                   10    UNLINK             259             0       0
                   11    TERMINATE          259             0       0
                   12    GENERATE             1             0       0
                   13    TERMINATE            1             0       0


FACILITY         ENTRIES  UTIL.   AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY
 SRV                260    0.996      1.915  1     1017    0    0     0     0


USER CHAIN          SIZE RETRY  AVE.CONT    ENTRIES  MAX     AVE.TIME
 LINE                761    0    376.896      1020   761      184.753


FEC XN   PRI          BDT      ASSEM  CURRENT  NEXT  PARAMETER     VALUE
  1017    0         500.173    1017      8       9
                                                     PTIME         2.064

  1024    0         501.041    1024      0       1
  1022    0         501.045    1022      0       4
  1025    0        1000.000    1025      0      12
```

Figure 7. Output of simulation in GPSS

3. MoE statistics

Table 3. Model 2 MoE statistics

Random seed	*Moe 1 - Downtime factor*	*MoE 2 – Average of all jobs in queue*
1	0.996	0
2	0.998	0

3	0.997	0
4	0.999	0
5	0.996	0
6	0.998	0
7	0.998	0
8	1	0
9	0.997	0
10	0.998	0
Statistics	Average: 0.9977 Mode: 0.998 Median: 0.998 Standard deviation: 0.001 95% interval: 0.997 to 1	Average: 0 Mode: 0 Median: 0 Standard deviation: 0 95% interval: 0 to 0

VI MODEL 3 DEVELOPMENT: ANYLOGIC

Model is written in AnyLogic 8.8.6 Personal Learning Edition. The source code for the program is available in the Model3.alp file located in the Model 3 folder or ANNEX bellow.

1. Description

There are four categories of elements (agents) in the model:

- Source: the job's type-specific entry point. [J1 and J2]
- Queue: a buffer used to hold jobs until the server is ready to handle them. [queue]
- Delay: a processing tool that "delays" the movement of a work from the queue to the exit. [Server]
- Sink: the place where all jobs end. [sink]

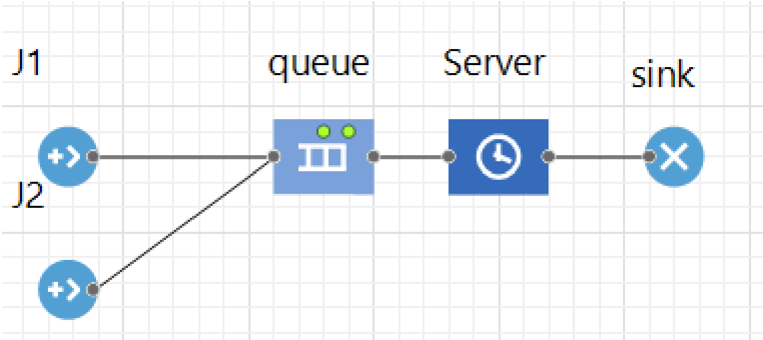

Figure 8. System architecture in AnyLogic

AnyLogic elements are configurated:

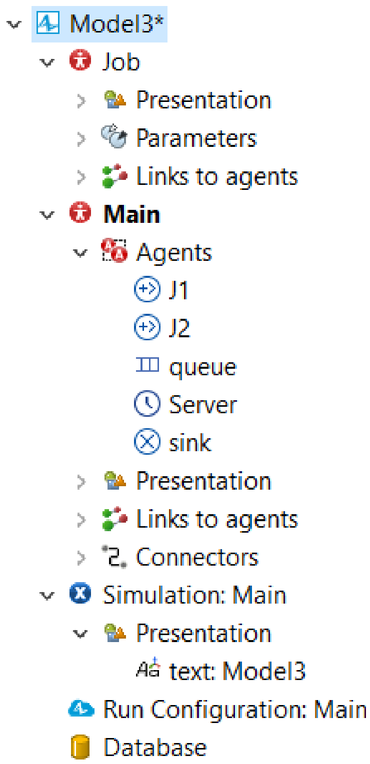

Figure 9. Project field

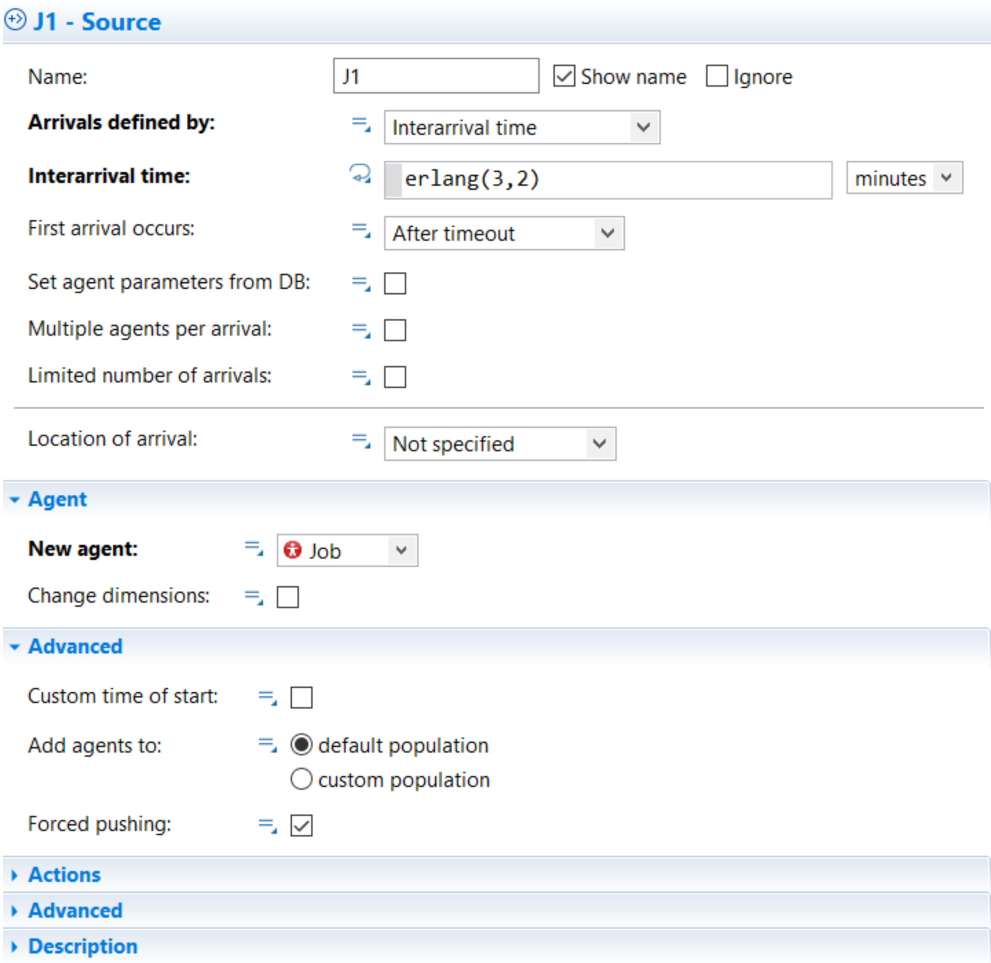

Figure 10.J1 configuration in AnyLogic GUI

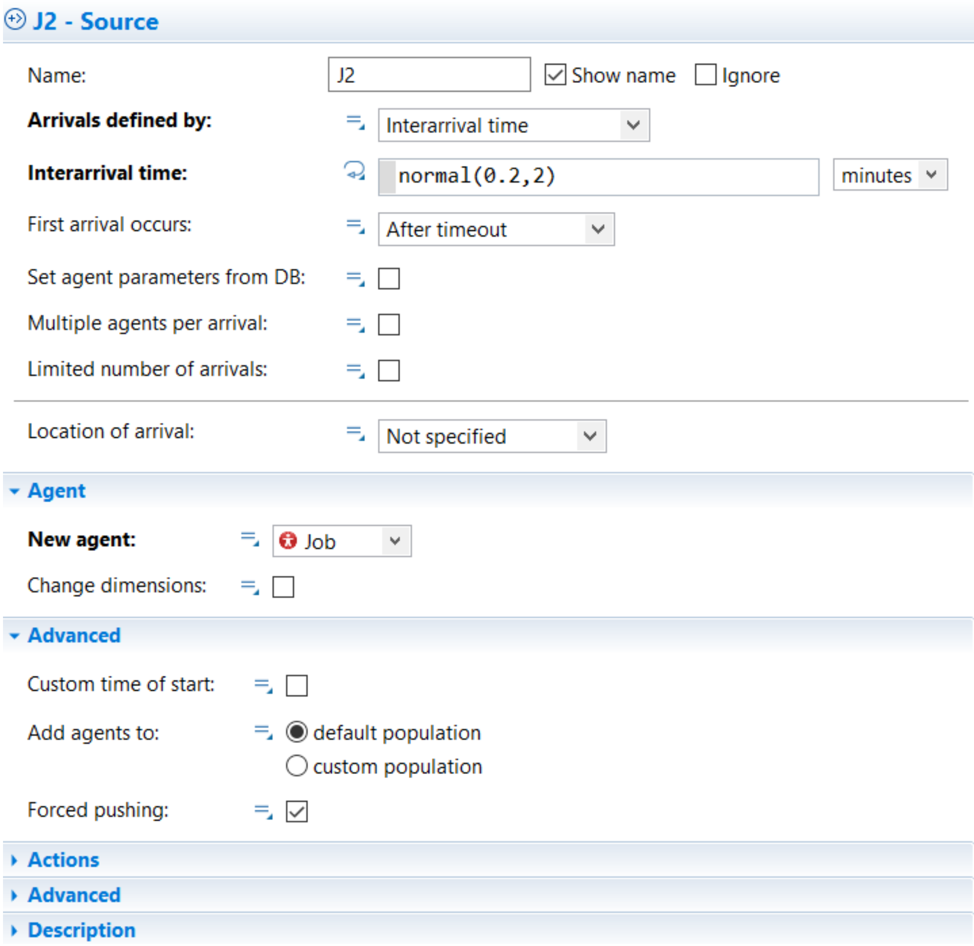

Figure 11.J2 configuration in AnyLogic GUI

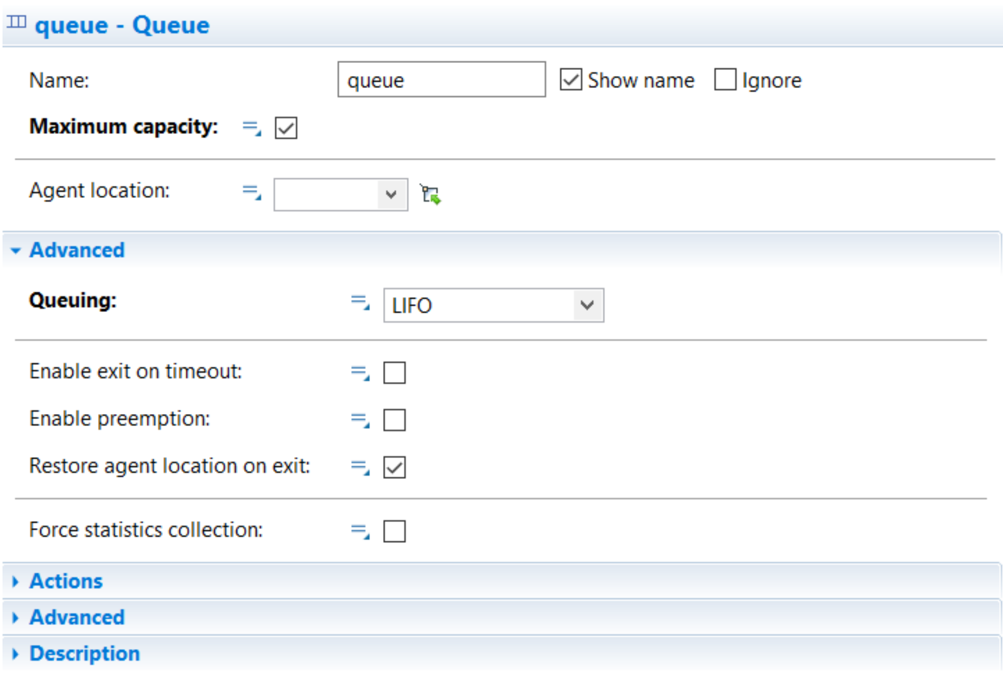

Figure 12. Queue configuration in AnyLogic GUI

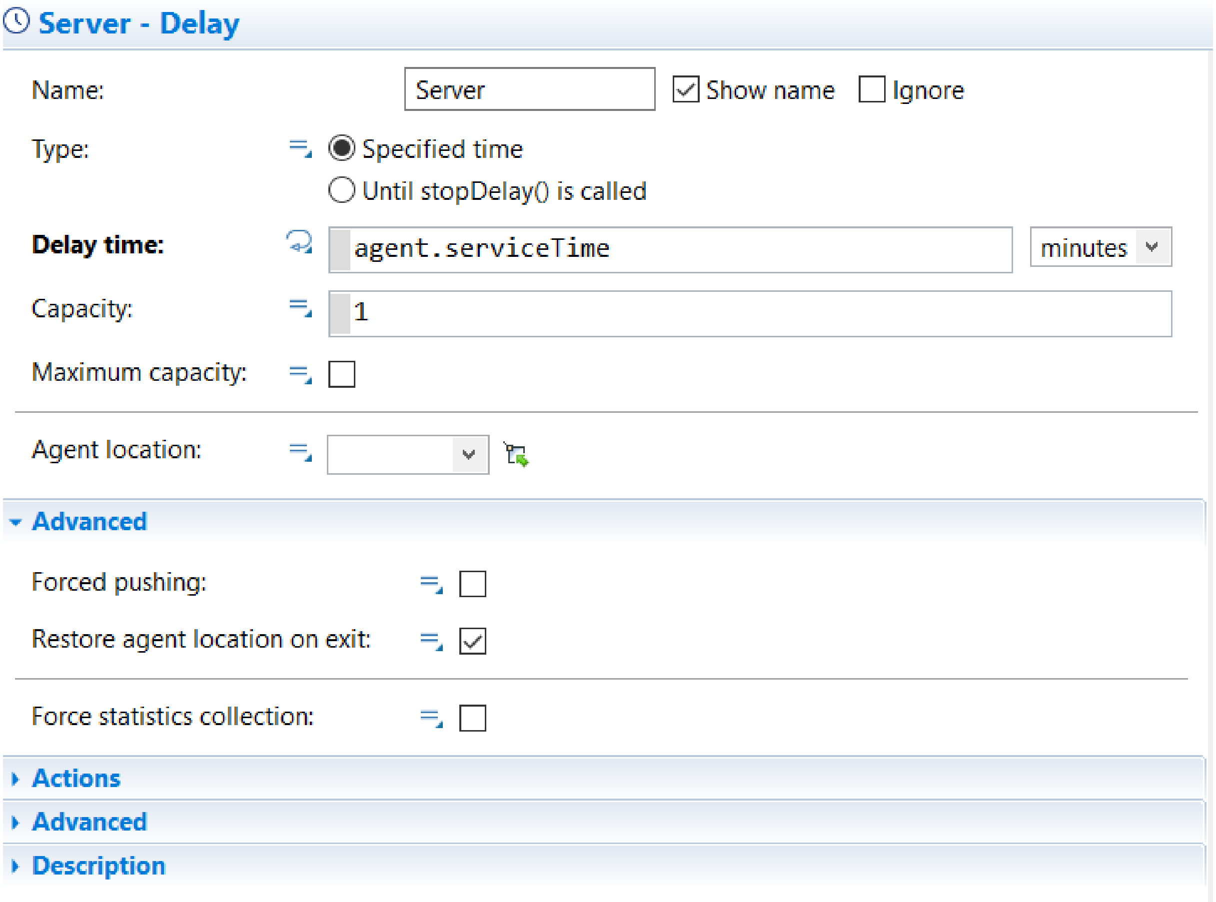

Figure 13. Server configuration in AnyLogic GUI

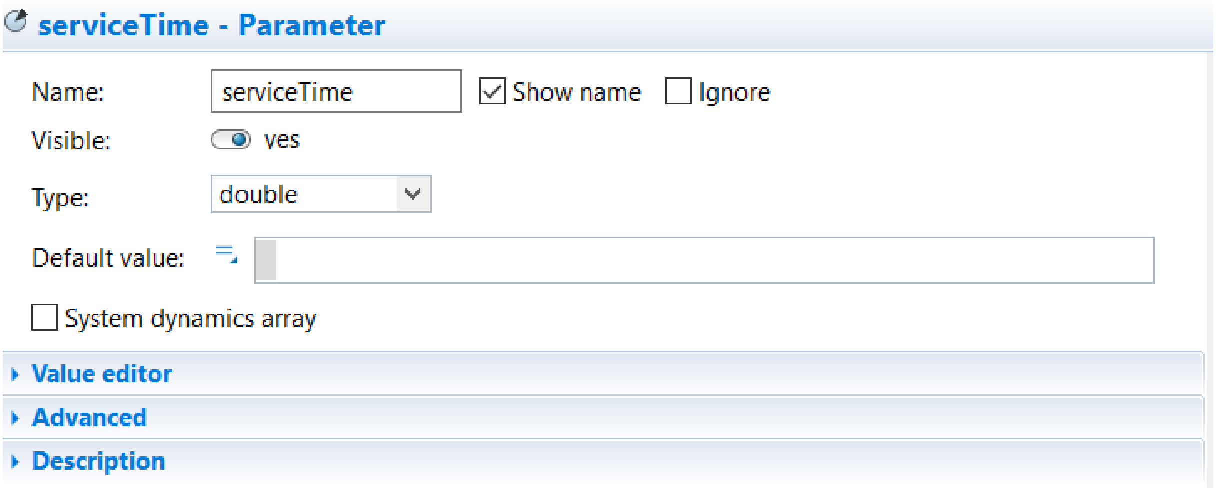

Figure 14. Custom parameter in "Job" agent configuration in AnyLogic GUI

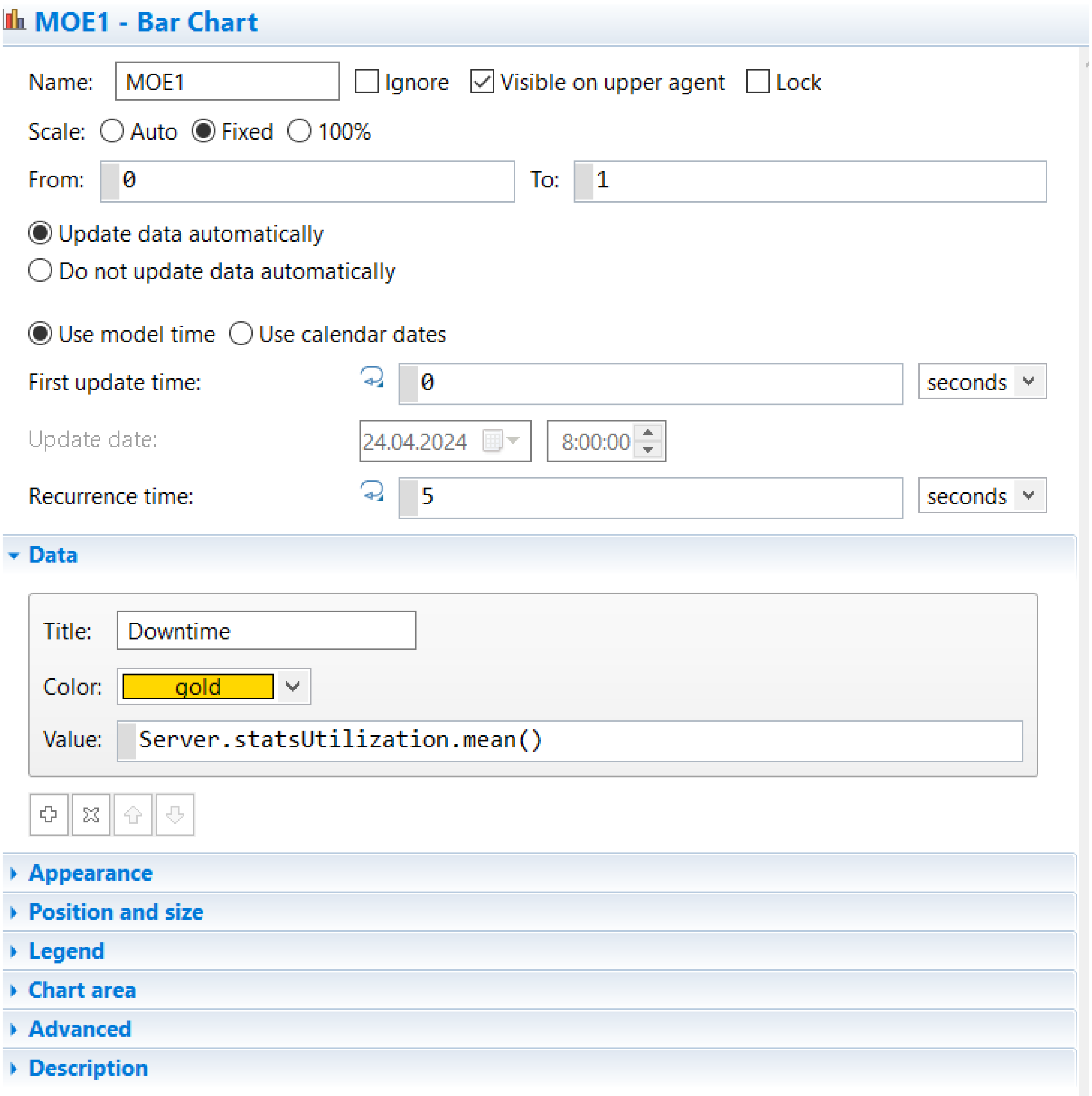

Figure 15. MOE1 bar configuration in AnyLogic GUI

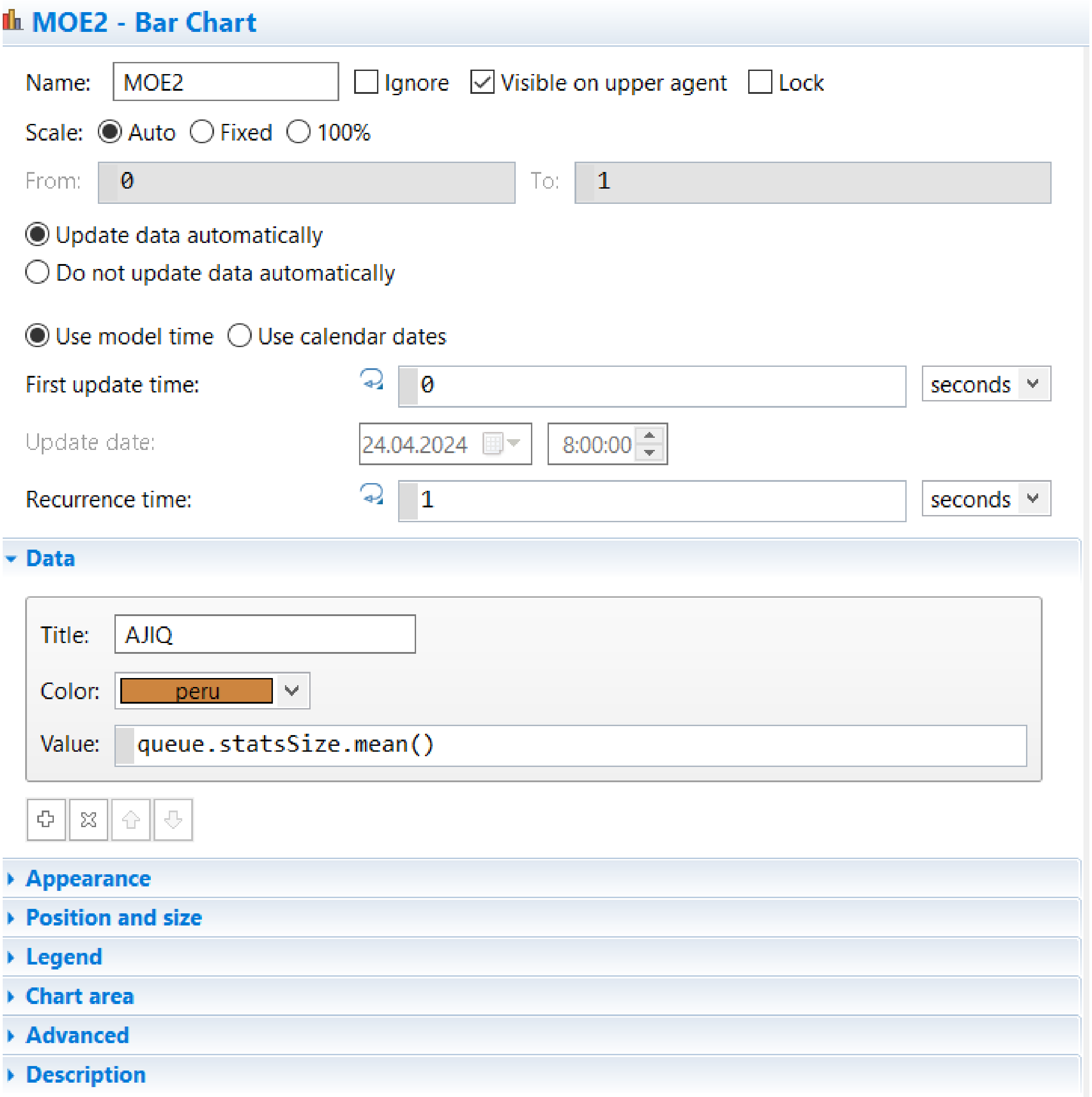

Figure 16. MOE2 bar configuration in AnyLogic GUI

2. Simulation results

Random seed:78727

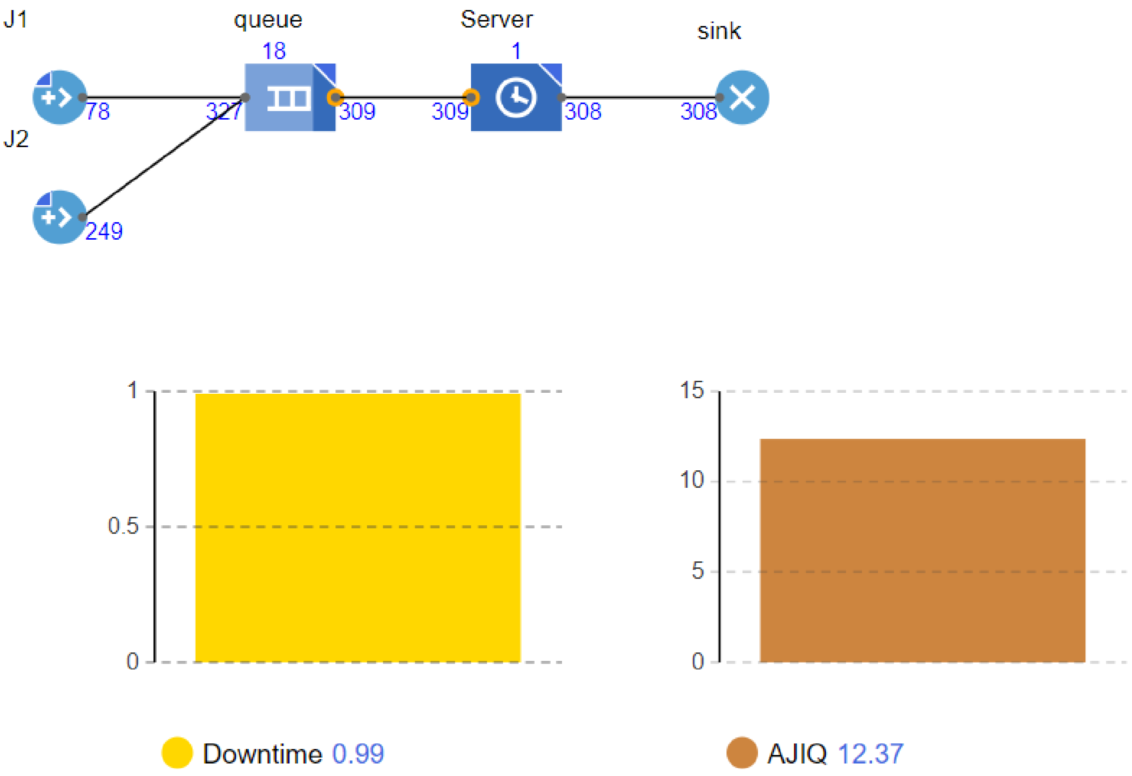

Figure 17. Model 3 simulation results

3. MoE statistics

Table 4. Model 3 MoE statistics

Random seed	*Moe 1 - Downtime factor*	*MoE 2 – Average of all jobs in queue*
1	0.99	15.9
2	1	17.02
3	0.99	15.86
4	0.98	18.44
5	0.99	15
6	0.97	18.45
7	0.99	9.94
8	1	23.07
9	0.99	15.94
10	0.99	14.32
Statistics	Average: 0.99 Mode: 0.99 Median: 0.99 Standard deviation: 0.01 95% interval: 0.97 to 1	Average: 16.39 Mode: 15 Median: 15.92 Standard deviation: 3.19 95% interval: 10.93 to 22.03

VII SIMULATION RESULTS COMPARISON

Table 5. Results Comparison

Random seed	*MoE1(1)*	*MoE2(2)*	*Moe1(2)*	*MoE2(2)*	*MoE1(3)*	*MoE2(3)*
1	0.69	0	0.996	0	0.99	15.9
2	0.678	0	0.998	0	1	17.02
3	0.706	0	0.997	0	0.99	15.86
4	0.68	0	0.999	0	0.98	18.44
5	0.694	0	0.996	0	0.99	15
6	0.678	0	0.998	0	0.97	18.45
7	0.686	0	0.998	0	0.99	9.94
8	0.69	0	1	0	1	23.07
9	0.684	0	0.997	0	0.99	15.94
10	0.698	0	0.998	0	0.99	14.32
Statistics	Average: 0.689 Mode: 0.69 Median: 0.688 Standard deviation: 0.001 95% interval: 0.678 to 0.704	Average: 0 Mode: 0 Median: 0 Standard deviation: 0 95% interval: 0 to 0	Average: 0.9977 Mode: 0.998 Median: 0.998 Standard deviation: 0.001 95% interval: 0.997 to 1	Average: 0 Mode: 0 Median: 0 Standard deviation: 0 95% interval: 0 to 0	Average: 0.99 Mode: 0.99 Median: 0.99 Standard deviation: 0.01 95% interval: 0.97 to 1	Average: 16.39 Mode: 15 Median: 15.92 Standard deviation: 3.19 95% interval: 10.93 to 22.03

Several reasons can be attributed to the observed variances in simulation outcomes across different environments:

Modelling Approach: Because various simulation environments use different modelling approaches and algorithms, the outcomes might vary.

Implementation Details: Variations in the way that event processing, random number generation, and other simulation components are handled might affect the results.

System Configuration: A simulation's outcome may be affected by the way its parameters—such as service times, arrival rates, and queue management techniques—are set up.

It's also important to keep in mind that variations in the simulation's starting circumstances, such the existence of negative timers, could have had an impact on the variations in outcomes between the Python-based simulation and other simulations.

VIII CONCLUSIONS

The considerable influence of simulation platforms on modelling outputs is demonstrated by the comparison of simulation results across Python, GPSS, and Anylogic environments. When evaluating simulation findings, care must be taken to account for variations in implementation details, modelling approaches, and system complexity. To guarantee the accuracy and applicability of simulation results in real-world settings, researchers and practitioners should give careful validation, sensitivity analysis, and consideration of context-specific elements top priority going ahead.

In my work, you can see that I had difficulty with model 1, since the results of AnyLogic and GPSS were different initially. It also bothered me and took a lot of time to work with a negative timer, after which it turned out that I needed to change the std to something else and the error disappeared

IX ANNEX

1.model1

```
import numpy as np
import csv
from collections import deque

# RNG for exponential distribution
class CustomRandomGenerator:
    def __init__(self, seed=None):
        self.seed = seed
        self.a = 1664525
        self.c = 1013904223
        self.m = 2 ** 32
        self.current = seed

    def rand(self):
        self.current = (self.a * self.current + self.c) % self.m
        return self.current / self.m

def erlang(rng, k, lambd):
    result = 0
    for _ in range(k):
        result += -np.log(1 - rng.rand()) / lambd
    return result

def normal(mean, std_dev, rng):
    u1 = rng.rand()
    u2 = rng.rand()
    z = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    return mean + std_dev * z

# Define model parameters
lambda1 = 3
lambda2 = 2
mu1 = 2
mu2 = 1.5
service_time1 = 0
service_time2 = 0
server_status = 0  # 0 - free, 1 - busy
queue = deque()  # Task queue (stack)
current_time = 0  # Current time
simulation_time = 501  # Simulation time
rng = CustomRandomGenerator(seed=78727)  # Random number generator

# Statistics for calculating MOE1 and MOE2
downtime_times = []  # Server downtime times
queue_lengths = []  # Queue lengths at different time points

# Function to print current simulation status
def print_simulation_status(event_type, current_time, j1, j2,
server_status, queue):
    print("Time:", current_time, "Event:", event_type, "J1:", j1 if j1 !=
float('inf') else "-", "J2:",
          j2 if j2 != float('inf') else "-", "Server status:", "Busy" if
server_status == 1 else "Free",
          "Queue length:", len(queue))

# Start simulation
print("Simulation started.")
print_simulation_status("Start", current_time, "-", "-", server_status,
queue)
```

```
# Simulation events
simulation_data = []  # For saving simulation data
while current_time < simulation_time:
    # Generate time until arrival of next task for each type
    interarrival_time1 = erlang(rng, 2, lambda1)
    interarrival_time2 = np.random.normal(2, 0.2)
    j1 = current_time + interarrival_time1 if current_time +
interarrival_time1 < simulation_time else float('inf')
    j2 = current_time + interarrival_time2 if current_time +
interarrival_time2 < simulation_time else float('inf')

    # Determine service time for the current task
    if server_status == 0:
        service_time1 = np.random.normal(2, 0.15)
        service_time2 = np.random.normal(1.5, 0.5)

    # Choose the event that occurs first
    min_time = min(j1, j2, service_time1, service_time2)
    event_type = None
    if j1 == min_time:
        event_type = "J1"
    elif j2 == min_time:
        event_type = "J2"
    elif service_time1 == min_time:
        event_type = "E1"
    elif service_time2 == min_time:
        event_type = "E2"

    # Update current time
    current_time += min_time

    # Recalculate interarrival and service times based on the updated
current time
    if j1 != float('inf'):
        j1 -= min_time
    if j2 != float('inf'):
        j2 -= min_time
    if server_status == 1:
        service_time1 -= min_time
        service_time2 -= min_time

    # Update server status and queue
    if event_type == "J1":
        queue.appendleft("J1")  # Add to the front of the queue
    elif event_type == "J2":
        queue.appendleft("J2")  # Add to the front of the queue

    # Service tasks in the queue if the server is free
    if server_status == 0:
        if queue:
            server_status = 1
            job_type = queue.pop()  # Remove the last element from the
queue
            if job_type == "J1":
                service_time1 = np.random.normal(2, 1.5)  # Service time
for Type J1 tasks
            elif job_type == "J2":
                service_time2 = np.random.normal(1.5, 0.5)  # Service time
for Type J2 tasks
    elif server_status == 1:
        if not queue:
            server_status = 0
```

```python
    # Update queue if server is busy and tasks are still arriving
    elif server_status == 1:
        if event_type == "J1" or event_type == "J2":
            queue.appendleft(event_type)  # Add to the front of the queue
            queue_lengths.append(len(queue))  # Update queue length
            # Update interarrival times for tasks in queue
            for i in range(len(queue)):
                if queue[i] == "J1":
                    j1 -= min_time
                elif queue[i] == "J2":
                    j2 -= min_time

    # Save data for statistics
    if server_status == 0:
        downtime_times.append(current_time)
    queue_lengths.append(len(queue))

    # Print current simulation status
    print_simulation_status(event_type, current_time, j1, j2,
server_status, queue)

    # Save simulation data
    simulation_data.append(
        [current_time, event_type, j1, j2, "Busy" if server_status == 1
else "Free", len(queue), str(queue)])

# Print table with simulation results
print("\nSimulation process table template")
print("{:<2} {:<2} {:<2} {:<2} {:<2} {:<2} {:<2}".format("Num", "Time",
"Event", "J1", "J2", "St", "S", "n", "Q"))

# Print first 10 rows
for i, row in enumerate(simulation_data[:10]):
    print("{:<2} {:<2} {:<2} {:<2} {:<2} {:<2} {:<2}".format(i + 1, *row))

# Print last 10 rows
print("...")
for i, row in enumerate(simulation_data[-10:], start=len(simulation_data) -
9):
    print("{:<2} {:<2} {:<2} {:<2} {:<2} {:<2} {:<2}".format(i, *row))

# Calculate MOE1 and MOE2 metrics
downtime_factor = len(downtime_times) / simulation_time
average_jobs_in_queue = sum(queue_lengths) / len(queue_lengths)

# Print MOE1 and MOE2 values
print("\nMOE1 Downtime factor:", downtime_factor)
print("MOE2 Average of jobs in queue:", average_jobs_in_queue)

# Save simulation data to a CSV file
with open('simulation_results.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["Time", "Event", "J1", "J2", "Server status", "Queue
length", "Queue"])
    writer.writerows(simulation_data)

print("\nSimulation results have been saved to simulation_results.csv")
```

rand.py

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd
## example: https://stackoverflow.com/questions/56985271/linear-
```

```
congruential-generator-how-to-choose-seeds-and-statistical-tests ###
# RNG for exponential distribution
class CustomRandomGenerator:
    def __init__(self, seed=None):
        self.seed = seed
        self.a = 1664525
        self.c = 1013904223
        self.m = 2**32
        self.current = seed

    def rand(self):
        self.current = (self.a * self.current + self.c) % self.m
        return self.current / self.m

def custom_erlang(k, theta, rng):
    result = 0
    for _ in range(k):
        result += -theta * np.log(1 - rng.rand())
    return result

# Initialize the custom random number generator
rng = CustomRandomGenerator(seed=78727)

# Generate samples from Erlang distribution
sample_size = 1000
k = 2
theta = 3
sample = [custom_erlang(k, theta, rng) for _ in range(sample_size)]

# Plot histogram for Erlang distribution
plt.figure(figsize=(8, 6))
plt.hist(sample, bins=30, density=True, color='skyblue', edgecolor='black',
alpha=0.7, label='Empirical Distribution')
plt.hist(stats.gamma.rvs(a=k, scale=theta, size=sample_size), bins=30,
density=True, color='red', edgecolor='black', alpha=0.5, label='Theoretical
PDF')
plt.title('Erlang Distribution Comparison')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend()
plt.show()

# Perform hypothesis testing
# Kolmogorov-Smirnov test
ks_stat, ks_p = stats.kstest(sample, 'gamma', args=(k, 0, theta))

# Calculate mean for empirical distribution
mean_empirical = np.mean(sample)

# Calculate theoretical mean
mean_theoretical = k * theta

# Calculate shape and scale parameters for theoretical distribution
shape_theoretical, _, scale_theoretical = stats.gamma.fit(sample, floc=0)

# Define data for the table
data = {
    'SHAPE (K)': [shape_theoretical, k],
    'SCALE (THETA)': [scale_theoretical, theta]
}

# Create DataFrame
df = pd.DataFrame(data, index=['EMPIRICAL', 'THEORETICAL'])
print(df)
```

```
# Print test results
alpha = 0.05
if ks_p < alpha:
    print('Kolmogorov-Smirnov test: statistic = {}, p-value = 
{}'.format(ks_stat, ks_p))
    print('Reject the null hypothesis')
else:
    print('Kolmogorov-Smirnov test: statistic = {}, p-value = 
{}'.format(ks_stat, ks_p))
    print('Fail to reject the null hypothesis')


observed_freq, _ = np.histogram(sample, bins=30, density=True)
chi2_stat_1, chi2_p_1 = stats.chisquare(observed_freq)
alpha = 0.05
if chi2_p_1 < alpha:
    print('Chi-square test: statistic = {}, p-value = 
{}'.format(chi2_stat_1, chi2_p_1))
    print('Reject the null hypothesis')
else:
    print('Chi-square test: statistic = {}, p-value = 
{}'.format(chi2_stat_1, chi2_p_1))
    print('Fail to reject the null hypothesis')
```

rand nornal.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd

# RNG for exponential distribution
class CustomRandomGenerator:
    def __init__(self, seed=None):
        self.seed = seed
        self.a = 1664525
        self.c = 1013904223
        self.m = 2**32
        self.current = seed

    def rand(self):
        self.current = (self.a * self.current + self.c) % self.m
        return self.current / self.m

def custom_normal(mean, std_dev, rng):
    u1 = rng.rand()
    u2 = rng.rand()
    z = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    return mean + std_dev * z

# Initialize the custom random number generator
rng = CustomRandomGenerator(seed=42)

# Generate samples from Normal distribution
sample_size = 1000
mean = 2
std_dev = 0.2
sample = [custom_normal(mean, std_dev, rng) for _ in range(sample_size)]

# Plot histogram for normal distribution
plt.figure(figsize=(8, 6))
plt.hist(sample, bins=30, density=True, color='skyblue', edgecolor='black', 
alpha=0.7, label='Empirical Distribution')
plt.hist(np.random.normal(mean, std_dev, size=sample_size), bins=30, 
```

```
density=True, color='red', edgecolor='black', alpha=0.5, label='Theoretical
PDF')
plt.title('Normal Distribution Comparison')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend()
plt.show()

# Calculate mean and standard deviation for empirical distribution
mean_empirical = np.mean(sample)
std_dev_empirical = np.std(sample)

# Calculate theoretical mean and standard deviation
mean_theoretical = mean
std_dev_theoretical = std_dev

# Create DataFrame for the table
data = {
    'MEAN': [mean_empirical, mean_theoretical],
    'STANDARD DEVIATION': [std_dev_empirical, std_dev_theoretical]
}

df = pd.DataFrame(data, index=['EMPIRICAL', 'THEORETICAL'])
print(df)

# Perform hypothesis testing
# Kolmogorov-Smirnov test
ks_stat, ks_p = stats.kstest(sample, 'norm', args=(mean, std_dev))
alpha = 0.05
if ks_p < alpha:
    print('Kolmogorov-Smirnov test: statistic = {}, p-value =
{}'.format(ks_stat, ks_p))
    print('Reject the null hypothesis')
else:
    print('Kolmogorov-Smirnov test: statistic = {}, p-value =
{}'.format(ks_stat, ks_p))
    print('Fail to reject the null hypothesis')

# Chi-square test
observed_freq, _ = np.histogram(sample, bins=30, density=True)
chi2_stat_1, chi2_p_1 = stats.chisquare(observed_freq)
alpha = 0.05
if chi2_p_1 < alpha:
    print('Chi-square test: statistic = {}, p-value =
{}'.format(chi2_stat_1, chi2_p_1))
    print('Reject the null hypothesis')
else:
    print('Chi-square test: statistic = {}, p-value =
{}'.format(chi2_stat_1, chi2_p_1))
    print('Fail to reject the null hypothesis')
```

rand normal1.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd

# RNG for exponential distribution
class CustomRandomGenerator:
    def __init__(self, seed=None):
        self.seed = seed
        self.a = 1664525
```

```
        self.c = 1013904223
        self.m = 2**32
        self.current = seed

    def rand(self):
        self.current = (self.a * self.current + self.c) % self.m
        return self.current / self.m

def custom_normal(mean, std_dev, rng):
    u1 = rng.rand()
    u2 = rng.rand()
    z = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    return mean + std_dev * z

# Initialize the custom random number generator
rng = CustomRandomGenerator(seed=42)

# Generate samples from Normal distribution
sample_size = 1000
mean = 2
std_dev = 1.5
sample = [custom_normal(mean, std_dev, rng) for _ in range(sample_size)]

# Plot histogram for normal distribution
plt.figure(figsize=(8, 6))
plt.hist(sample, bins=30, density=True, color='skyblue', edgecolor='black',
alpha=0.7, label='Empirical Distribution')
plt.hist(np.random.normal(mean, std_dev, size=sample_size), bins=30,
density=True, color='red', edgecolor='black', alpha=0.5, label='Theoretical
PDF')
plt.title('Normal Distribution Comparison')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend()
plt.show()

# Calculate mean and standard deviation for empirical distribution
mean_empirical = np.mean(sample)
std_dev_empirical = np.std(sample)

# Calculate theoretical mean and standard deviation
mean_theoretical = mean
std_dev_theoretical = std_dev

# Create DataFrame for the table
data = {
    'MEAN': [mean_empirical, mean_theoretical],
    'STANDARD DEVIATION': [std_dev_empirical, std_dev_theoretical]
}

df = pd.DataFrame(data, index=['EMPIRICAL', 'THEORETICAL'])
print(df)

# Perform hypothesis testing
# Kolmogorov-Smirnov test
ks_stat, ks_p = stats.kstest(sample, 'norm', args=(mean, std_dev))
alpha = 0.05
if ks_p < alpha:
    print('Kolmogorov-Smirnov test: statistic = {}, p-value =
{}'.format(ks_stat, ks_p))
    print('Reject the null hypothesis')
else:
    print('Kolmogorov-Smirnov test: statistic = {}, p-value =
{}'.format(ks_stat, ks_p))
```

```
    print('Fail to reject the null hypothesis')

# Chi-square test
observed_freq, _ = np.histogram(sample, bins=30, density=True)
chi2_stat_1, chi2_p_1 = stats.chisquare(observed_freq)
alpha = 0.05
if chi2_p_1 < alpha:
    print('Chi-square test: statistic = {}, p-value =
{}'.format(chi2_stat_1, chi2_p_1))
    print('Reject the null hypothesis')
else:
    print('Chi-square test: statistic = {}, p-value =
{}'.format(chi2_stat_1, chi2_p_1))
    print('Fail to reject the null hypothesis')
```

rand normal2.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd

# RNG for exponential distribution
class CustomRandomGenerator:
    def __init__(self, seed=None):
        self.seed = seed
        self.a = 1664525
        self.c = 1013904223
        self.m = 2**32
        self.current = seed

    def rand(self):
        self.current = (self.a * self.current + self.c) % self.m
        return self.current / self.m

def custom_normal(mean, std_dev, rng):
    u1 = rng.rand()
    u2 = rng.rand()
    z = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    return mean + std_dev * z

# Initialize the custom random number generator
rng = CustomRandomGenerator(seed=42)

# Generate samples from Normal distribution
sample_size = 1000
mean = 1.5
std_dev = 0.5
sample = [custom_normal(mean, std_dev, rng) for _ in range(sample_size)]

# Plot histogram for normal distribution
plt.figure(figsize=(8, 6))
plt.hist(sample, bins=30, density=True, color='skyblue', edgecolor='black',
alpha=0.7, label='Empirical Distribution')
plt.hist(np.random.normal(mean, std_dev, size=sample_size), bins=30,
density=True, color='red', edgecolor='black', alpha=0.5, label='Theoretical
PDF')
plt.title('Normal Distribution Comparison')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend()
plt.show()
```

```
# Calculate mean and standard deviation for empirical distribution
mean_empirical = np.mean(sample)
std_dev_empirical = np.std(sample)

# Calculate theoretical mean and standard deviation
mean_theoretical = mean
std_dev_theoretical = std_dev

# Create DataFrame for the table
data = {
    'MEAN': [mean_empirical, mean_theoretical],
    'STANDARD DEVIATION': [std_dev_empirical, std_dev_theoretical]
}

df = pd.DataFrame(data, index=['EMPIRICAL', 'THEORETICAL'])
print(df)

# Perform hypothesis testing
# Kolmogorov-Smirnov test
ks_stat, ks_p = stats.kstest(sample, 'norm', args=(mean, std_dev))
alpha = 0.05
if ks_p < alpha:
    print('Kolmogorov-Smirnov test: statistic = {}, p-value =
{}'.format(ks_stat, ks_p))
    print('Reject the null hypothesis')
else:
    print('Kolmogorov-Smirnov test: statistic = {}, p-value =
{}'.format(ks_stat, ks_p))
    print('Fail to reject the null hypothesis')

# Chi-square test
observed_freq, _ = np.histogram(sample, bins=30, density=True)
chi2_stat_1, chi2_p_1 = stats.chisquare(observed_freq)
alpha = 0.05
if chi2_p_1 < alpha:
    print('Chi-square test: statistic = {}, p-value =
{}'.format(chi2_stat_1, chi2_p_1))
    print('Reject the null hypothesis')
else:
    print('Chi-square test: statistic = {}, p-value =
{}'.format(chi2_stat_1, chi2_p_1))
    print('Fail to reject the null hypothesis')
```