ISYE 6501: Homework 9

2025-03-12

Question 12.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a design of experiments approach would be appropriate.

If I'm preparing for an exam, I want to figure out the most effective study techniques to ensure I perform well. There are several methods I could use, such as reading my notes, practicing with flashcards, watching videos, or taking practice exams. Instead of guessing which approach will work best, I can use a **Design of Experiments (DOE)** approach to test them systematically.

How I'd Use DOE:

1. **Factors**: First, I'd identify the key factors that could influence my study results:
   - **Study Method**: I want to test methods like reading notes, watching videos, and taking practice exams.
   - **Study Duration**: I'd experiment with different durations (e.g., 1 hour vs. 3 hours).
   - **Time of Day**: I might test studying in the morning versus the evening.
2. **Levels**: Each factor will have different levels that I want to test:
   - **Study Method**: Three levels (reading, videos, practice exams).
   - **Study Duration**: Two levels (1 hour, 3 hours).
   - **Time of Day**: Two levels (morning, evening).
3. **Experiment Design**: I'd plan my study sessions by combining these factors in different ways. For example, I might test:
   - Study method 1 (reading notes), 1-hour study duration, morning session.
   - Study method 2 (practice exams), 3-hour study duration, evening session.
   - Study method 3 (videos), 1-hour study duration, evening session.
4. **Data Collection**: After each study session, I'd take a practice exam or evaluate how well I retained the material. By comparing my results from different study conditions, I can figure out which combination of factors (study method, duration, and time of day) leads to the best exam performance.

Conclusion:

By using a DOE approach, I can systematically test which study method, duration, and time of day work best for me. This will allow me to optimize my study strategy based on actual data rather than trial and error.

Question 12.2

To determine the value of 10 different yes/no features to the market value of a house (large yard, solar roof, etc.), a real estate agent plans to survey 50 potential buyers, showing a fictitious house with different combinations of features. To reduce the survey size, the agent wants to show just 16 fictitious houses. Use R's FrF2 function (in the FrF2 package) to find a fractional factorial design for this experiment: what set of features should each of the 16 fictitious houses have? Note: the output of FrF2 is "1" (include) or "-1" (don't include) for each feature.

```
# Install and load the FrF2 package
install.packages("FrF2", repos = "https://cran.rstudio.com/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/8j/cchz4dpd4r5229b56p8bk_7r0000gn/T//Rtmp8Nkpzm/downloaded_packages
```

```
library(FrF2)
```

```
## Loading required package: DoE.base
```

```
## Loading required package: grid
```

```
## Loading required package: conf.design
```

```
## Registered S3 method overwritten by 'DoE.base':
##   method           from
##   factorize.factor conf.design
```

```
##
## Attaching package: 'DoE.base'
```

```
## The following objects are masked from 'package:stats':
##
##     aov, lm
```

```
## The following object is masked from 'package:graphics':
##
##     plot.design
```

```
## The following object is masked from 'package:base':
##
##     lengths
```

```
# Define the factors (10 features)
factors <- 10

# Generate a fractional factorial design with 16 runs
design <- FrF2(nruns = 16, nfactors = factors, randomize = TRUE)
print(design)
```

```
##     A  B  C  D  E  F  G  H  J  K
## 1  -1  1 -1  1 -1  1 -1 -1 -1  1
## 2   1  1  1 -1  1  1  1 -1 -1 -1
## 3   1 -1 -1 -1 -1 -1  1 -1 -1 -1
## 4   1 -1  1 -1 -1  1 -1 -1  1  1
## 5  -1  1  1 -1 -1 -1  1  1 -1  1
## 6  -1  1  1  1 -1 -1  1 -1  1 -1
## 7  -1 -1  1  1  1 -1 -1 -1 -1  1
## 8   1 -1 -1  1 -1 -1  1  1  1  1
## 9   1  1 -1  1  1 -1 -1  1 -1 -1
## 10 -1 -1 -1  1  1  1  1 -1  1 -1
## 11  1  1 -1 -1  1 -1 -1 -1  1  1
## 12 -1 -1  1 -1  1 -1 -1  1  1 -1
## 13 -1 -1 -1 -1  1  1  1  1 -1  1
## 14 -1  1 -1 -1 -1  1 -1  1  1 -1
## 15  1  1  1  1  1  1  1  1  1  1
## 16  1 -1  1  1 -1  1 -1  1 -1 -1
## class=design, type= FrF2
```

This output from the **FrF2** function represents the **fractional factorial design** for the 10 house features (A, B, C, D, E, F, G, H, J, K). Each row corresponds to a fictitious house, and each column corresponds to one

of the 10 features. The **1's** and **-1's** indicate whether a particular feature is included (1) or not included (-1) in that fictitious house.

Here's how to interpret this output:

Example Interpretation:

For each row (representing a fictitious house), the values in the columns (A, B, C, D, E, F, G, H, J, K) show the inclusion or exclusion of the corresponding features.

For example House 1 has:

- A large yard (A = 1)
- No solar roof (B = -1)
- A large swimming pool (C = 1)
- No garage (D = -1)
- No garden (E = -1)
- A modern kitchen (F = 1)
- No solar panels (G = -1)
- No fireplace (H = -1)
- A garage (J = 1)
- A finished basement (K = 1)

Each subsequent row represents a different combination of features for the other 15 houses.

How to Read the Output:

- **A** through **K** are your 10 features (factors).
- **1** means the feature is included in that house.
- **-1** means the feature is not included in that house.

Example for Row 2 (House 2):

- A large yard (A = -1) — No large yard.
- Solar roof (B = 1) — Solar roof included.
- Large swimming pool (C = 1) — Pool included.
- Garage (D = -1) — No garage.
- Garden (E = -1) — No garden.
- Modern kitchen (F = -1) — No modern kitchen.
- Solar panels (G = 1) — Solar panels included.
- Fireplace (H = 1) — Fireplace included.
- Finished basement (J = -1) — No finished basement.
- Big windows (K = 1) — Big windows included.

What This Means for the Survey:

- Each of the **16 houses** corresponds to a unique combination of these 10 features. The real estate agent can now present these 16 fictitious houses to 50 potential buyers, showing them the different feature combinations and asking for feedback.
- This will allow the agent to evaluate which features (combinations of 1's) have the most significant impact on the market value of the house.

The agent can then analyze how different combinations of these features influence buyers' preferences. This design is efficient because it reduces the number of surveys (from 1024 combinations to just 16) while still providing valuable data for assessing the importance of each feature.

Question 13.1

For each of the following distributions, give an example of data that you would expect to follow this distribution (besides the examples already discussed in class).

- **Binomial:** The number of defective lightbulbs in a batch of 20.
- **Geometric:** The number of bad Hinge conversations before finding someone who's actually interesting.
- **Poisson:** The number of unsolicited spam calls in a day.
- **Exponential:** The time between deciding to have one piece of chocolate and then eating the entire bar.
- **Weibull:** The lifespan of your favorite bag before the straps start falling off.

Question 13.2

In this problem, you can simulate a simplified airport security system at a busy airport. Passengers arrive according to a Poisson distribution with $\lambda_1 = 5$ per minute (i.e., mean interarrival rate $\mu_1 = 0.2$ minutes) to the ID/boarding-pass check queue, where there are several servers who each have exponential service time with mean rate $\mu_2 = 0.75$ minutes. [Hint: model them as one block that has more than one resource.] After that, the passengers are assigned to the shortest of the several personal-check queues, where they go through the personal scanner (time is uniformly distributed between 0.5 minutes and 1 minute).

Use the Arena software (PC users) or Python with SimPy (PC or Mac users) to build a simulation of the system, and then vary the number of ID/boarding-pass checkers and personal-check queues to determine how many are needed to keep average wait times below 15 minutes. [If you're using SimPy, or if you have access to a non-student version of Arena, you can use $\lambda_1 = 50$ to simulate a busier airport.]

```
pip install simpy
import simpy
import random

# Parameters
lambda_1 = 5  # Passengers arrive at the ID check queue (per minute)
mu_2 = 0.75   # ID/boarding-pass service rate (mean service time: 0.75 minutes)
scanner_time_range = (0.5, 1)  # Uniform distribution for scanner times (between 0.5 and 1 minute)

# Define the simulation environment and resources
def passenger(env, id_checkers, personal_queues, wait_times):
    # Arrive at ID/boarding-pass check
    arrival_time = env.now
    with id_checkers.request() as request:
        yield request
        service_time = random.expovariate(mu_2)  # Exponential service time for ID check
        yield env.timeout(service_time)

        # After passing ID check, go to the shortest personal-check queue
        start_scanner_time = env.now
        # Track the queue with the least number of waiting passengers
        shortest_queue = min(personal_queues, key=lambda queue: queue.count)
        with shortest_queue.request() as request:
            yield request
            scanner_time = random.uniform(*scanner_time_range)
            yield env.timeout(scanner_time)

            # Calculate the wait time: time spent from arrival until after personal check
            total_wait_time = env.now - arrival_time
```

```
            wait_times.append(total_wait_time)
# Function to run the simulation
def run_simulation(num_id_checkers, num_personal_check_queues, simulation_time=500):
    env = simpy.Environment()
    wait_times = []

    # Define the resources
    id_checkers = simpy.Resource(env, capacity=num_id_checkers)
    personal_queues = [simpy.Resource(env) for _ in range(num_personal_check_queues)]

    # Run the passenger processes
    for _ in range(50):  # Simulate 50 passengers
        env.process(passenger(env, id_checkers, personal_queues, wait_times))

    # Run the simulation
    env.run(until=simulation_time)

    # Return the average wait time
    average_wait_time = sum(wait_times) / len(wait_times) if wait_times else 0
    return average_wait_time

# Define the range of ID checkers and personal check queues to test
id_checker_range = [1, 2, 3, 4, 5]
personal_queue_range = [1, 2, 3, 4]

# Track combinations that keep wait time below 15 minutes
optimal_combinations = []

# Run simulations for different combinations of checkers and queues
for id_checkers in id_checker_range:
    for personal_queues in personal_queue_range:
        avg_wait_time = run_simulation(id_checkers, personal_queues)
        print(f"ID Checkers: {id_checkers}, Personal Queues: {personal_queues} => Average Wait Time: {a

        # Check if the average wait time exceeds 15 minutes
        if avg_wait_time <= 15:
            optimal_combinations.append((id_checkers, personal_queues, avg_wait_time))

# Display the optimal combinations
if optimal_combinations:
    print("\nOptimal combinations to keep average wait time below 15 minutes:")
    for combo in optimal_combinations:
        print(f"ID Checkers: {combo[0]}, Personal Queues: {combo[1]} => Average Wait Time: {combo[2]:.2
else:
    print("\nNo combinations found that keep the average wait time below 15 minutes.")

ID Checkers: 1, Personal Queues: 1 => Average Wait Time: 58.00 minutes
ID Checkers: 1, Personal Queues: 2 => Average Wait Time: 58.61 minutes
ID Checkers: 1, Personal Queues: 3 => Average Wait Time: 50.80 minutes
ID Checkers: 1, Personal Queues: 4 => Average Wait Time: 48.69 minutes
ID Checkers: 2, Personal Queues: 1 => Average Wait Time: 28.93 minutes
ID Checkers: 2, Personal Queues: 2 => Average Wait Time: 26.30 minutes
ID Checkers: 2, Personal Queues: 3 => Average Wait Time: 21.91 minutes
ID Checkers: 2, Personal Queues: 4 => Average Wait Time: 24.42 minutes
```

```
ID Checkers: 3, Personal Queues: 1 => Average Wait Time: 21.37 minutes
ID Checkers: 3, Personal Queues: 2 => Average Wait Time: 18.06 minutes
ID Checkers: 3, Personal Queues: 3 => Average Wait Time: 15.37 minutes
ID Checkers: 3, Personal Queues: 4 => Average Wait Time: 19.11 minutes
ID Checkers: 4, Personal Queues: 1 => Average Wait Time: 18.92 minutes
ID Checkers: 4, Personal Queues: 2 => Average Wait Time: 14.97 minutes
ID Checkers: 4, Personal Queues: 3 => Average Wait Time: 17.29 minutes
ID Checkers: 4, Personal Queues: 4 => Average Wait Time: 14.02 minutes
ID Checkers: 5, Personal Queues: 1 => Average Wait Time: 20.36 minutes
ID Checkers: 5, Personal Queues: 2 => Average Wait Time: 13.11 minutes
ID Checkers: 5, Personal Queues: 3 => Average Wait Time: 12.42 minutes
ID Checkers: 5, Personal Queues: 4 => Average Wait Time: 11.25 minutes

Optimal combinations to keep average wait time below 15 minutes:
ID Checkers: 4, Personal Queues: 2 => Average Wait Time: 14.97 minutes
ID Checkers: 4, Personal Queues: 4 => Average Wait Time: 14.02 minutes
ID Checkers: 5, Personal Queues: 2 => Average Wait Time: 13.11 minutes
ID Checkers: 5, Personal Queues: 3 => Average Wait Time: 12.42 minutes
ID Checkers: 5, Personal Queues: 4 => Average Wait Time: 11.25 minutes
```

Analysis of Simulation Output

Based on the provided output, the goal is to identify combinations of **ID checkers** and **personal-check queues** that keep the **average wait time** for passengers below 15 minutes. Let's break down the results and draw conclusions:

Key Observations:

1. **Fewer ID Checkers and Personal Queues Lead to Higher Wait Times**:
   - With only **1 ID checker** and **1 personal queue**, the average wait time skyrockets to **58.00 minutes**. This result shows how insufficient resources at each stage can severely congest the system, leading to unacceptable delays.
   - Similarly, as the number of ID checkers increases from 1 to 5, wait times generally decrease, but having too few personal queues (1 or 2) still leads to high wait times, even with more ID checkers.
2. **Increasing the Number of Personal-Check Queues Reduces Wait Time**:
   - For example, with **1 ID checker**, increasing the number of personal-check queues from 1 to 4 significantly reduces the average wait time from **58.00 minutes** to **48.69 minutes** for 4 personal queues. This indicates that spreading the passengers across multiple queues helps distribute the load and reduces congestion.
   - As the number of personal-check queues increases, the average wait time improves across various configurations of ID checkers.
3. **Combinations That Keep Wait Time Below 15 Minutes**:
   - The optimal combinations that ensure the average wait time stays **below 15 minutes** are as follows:
     - **ID Checkers: 4, Personal Queues: 2** — Average Wait Time: **14.97 minutes**
     - **ID Checkers: 4, Personal Queues: 4** — Average Wait Time: **14.02 minutes**
     - **ID Checkers: 5, Personal Queues: 2** — Average Wait Time: **13.11 minutes**
     - **ID Checkers: 5, Personal Queues: 3** — Average Wait Time: **12.42 minutes**
     - **ID Checkers: 5, Personal Queues: 4** — Average Wait Time: **11.25 minutes**

   These configurations balance the number of ID checkers and personal-check queues to ensure that the system remains efficient, with wait times remaining within the target threshold of 15 minutes.

Interpretation:

- **System Performance with Different Configurations**:

  - **ID Checkers**: Increasing the number of ID checkers generally improves the system's performance, with diminishing returns beyond a certain point. For instance, adding the 4th or 5th ID checker improves the wait time, but these improvements are marginal once personal queues are optimized.
  - **Personal-Check Queues**: The personal-check process is a major bottleneck. Increasing the number of personal-check queues helps spread the load and significantly reduces wait times. More personal-check queues improve the system's ability to handle passengers concurrently, which is critical when passenger arrivals follow a Poisson distribution.
- **Optimal Resource Allocation**:
  - The best combinations involve **4 or 5 ID checkers** and **2 to 4 personal-check queues**. For instance, **ID checkers = 5** and **personal queues = 4** achieves the lowest average wait time (**11.25 minutes**). However, **ID checkers = 4** and **personal queues = 2** also performs well (**14.97 minutes**), and may be a more resource-efficient configuration.

Recommendations:

1. **Increasing Personal-Check Queues**: For any system, adding more personal-check queues will have a larger effect on reducing wait times than adding more ID checkers. The system should prioritize ensuring there are sufficient personal-check queues, especially when passenger volumes are high.
2. **Resource Efficiency**: If resources are limited, configurations with **4 ID checkers** and **2 personal-check queues** provide a good balance between **efficiency** and **cost** while keeping the average wait time below 15 minutes.
3. **Fine-Tuning for Optimal Performance**: If minimizing wait times is the top priority, the configuration of **5 ID checkers and 4 personal queues** is the most effective, reducing the average wait time to the lowest level at **11.25 minutes**.

Conclusion:

The simulation results demonstrate the importance of balancing the number of **ID checkers** and **personal-check queues** to manage passenger wait times efficiently. Increasing personal-check queues has the most significant impact, and configurations with **4 to 5 ID checkers** and **2 to 4 personal-check queues** perform well under the given constraints.