Assignment 2

Prepared by Ali Younis & Manuel Martinez Garcia

Problem 1:

Link to LLM Conversation for Balking Logic

Link to GitHub Repo for more Detailed Work on Food Truck Simulator

Setting. Suppose that you run a food truck on each Wednesday between 11AM and 1PM. There is no customer waiting when the food truck opens at 11AM. Customers arrive at the food truck according to a Poisson process with rate λ persons per minute. There is a single line of queue with a first-come-first-serve rule. Customers that have arrived before 1PM will eventually get served. In other words, customers are not able to join the queue if they arrive later than 1PM. The food truck resumes its operation until all customers who have arrived before 1PM get served. You are the owner of the food truck and there is only one server (yourself). The service time for each customer is independent and identically distributed with some distribution F . The service time includes the time on taking orders and food preparation. The distribution F has a mean of $1/\mu$ and the probability distribution is to be specified. Suppose λ = 2. Simulate 100 independent days (Wednesdays) for each of the following parts.

To do. Simulate $n = 100$ independent days of operation from 11AM and 1PM and generate the sequence of customer waiting times. For this task, the service time distribution F is given by an exponential distribution with expectation as 35 seconds. Recall that the condition $\lambda = 2$ persons per minute in the problem statement implies that the exponentially distributed inter-arrival times of customers have expectation also as 30 seconds.

In [1]:
```python
import numpy as np
from FoodTruck import FoodTruck, format_time_from_minutes

### Default Parameters ###
LAMBDA = 2.0            # arrivals per minute
SERVICE_MEAN_SEC = 35    # average service time
T0 = 0                # 11 AM
T1 = 120              # 1 PM
N_DAYS = 100
```

a.) Suppose that you are interested in knowing that on a typical Wednesday, what is the expectation of the averaged waiting time for customers who arrive at the food truck between 11:15 AM to 11:30 AM. Compute an estimate of this quantity using simulation.

In [2]:
```python
AM_TOTAL_WAIT_TIMES = []
PM_TOTAL_WAIT_TIMES = []

for _ in range(N_DAYS):
    sim = FoodTruck(LAMBDA, SERVICE_MEAN_SEC, T0, T1)
    daily_arrivals, daily_waiting, _, _ = sim.simulate_day()
    # 11:15 - 11:30 AM arrivals
    AM_MASK = (daily_arrivals >= 15) & (daily_arrivals < 30)
    AM_WAIT_TIMES = daily_waiting[AM_MASK]
    AM_TOTAL_WAIT_TIMES.extend(AM_WAIT_TIMES)

    # 12:45 - 1:00 PM arrivals (for part b.)
    PM_MASK = (daily_arrivals >= 105) & (daily_arrivals < 120)
    PM_WAIT_TIMES = daily_waiting[PM_MASK]
    PM_TOTAL_WAIT_TIMES.extend(PM_WAIT_TIMES)

# Part (a)
AM_AVG_WAIT = np.mean(AM_TOTAL_WAIT_TIMES)
min, sec = format_time_from_minutes(AM_AVG_WAIT)
print(f"Average wait time for customers arriving between 11:15 and 11:30 AM over {N_DAYS} days: {min} minutes and {sec} seconds")
```

```
Average wait time for customers arriving between 11:15 and 11:30 AM over 100 days: 6 minutes and 11 seconds
```

b.) Suppose that you are interested in knowing that on a typical Wednesday, what is the expectation of the averaged waiting time for customers who arrive at the food truck between 12:45 PM to 1:00 PM. Compute an estimate of this quantity using simulation.

In [3]:
```
PM_AVG_WAIT = np.mean(PM_TOTAL_WAIT_TIMES)
min, sec = format_time_from_minutes(PM_AVG_WAIT)
print(f"Average wait time for customers arriving between 12:45 and 1:00 PM over {N_DAYS} days: {min} minutes and {sec} seconds")
```

```
Average wait time for customers arriving between 12:45 and 1:00 PM over 100 days: 22 minutes and 36 seconds
```

c.) Compare the two quantities you computed in the previous two questions. Describe intuition that you may get from this comparison.

> Wait times from later in the day are much larger. Almost by 15-20 minutes longer. This is probably because customers arrive at a faster rate that one chef is able to serve.

d.) Compute the percentage of customers who arrive at the food truck between 11:15 AM to 11:30 AM and wait for more than 3 minutes. Compute the percentage of customers who arrive at the food truck between 12:45 PM to 1:00 PM and wait for more than 3 minutes.

In [4]:
```
AM_over_3_min = np.sum(np.array(AM_TOTAL_WAIT_TIMES) > 3)
PM_over_3_min = np.sum(np.array(PM_TOTAL_WAIT_TIMES) > 3)
print(f"Proportion of customers waiting more than 3 minutes between 11:15 and 11:30 AM: {AM_over_3_min / len(AM_TOTAL_WAIT_TIMES):.2%}")
print(f"Proportion of customers waiting more than 3 minutes between 12:45 and 1:00 PM: {PM_over_3_min / len(PM_TOTAL_WAIT_TIMES):.2%}")
```

```
Proportion of customers waiting more than 3 minutes between 11:15 and 11:30 AM: 70.84%
Proportion of customers waiting more than 3 minutes between 12:45 and 1:00 PM: 98.08%
```

e.) Suppose that the customers will immediately abandon the system and leave for other dining options, conditional on that they see more than 5 people in the system (including the one being served). Now, what is the percentage of customers who abandon the system (=food truck) upon arrival between 12:45 PM and 1:00 PM? What is the expectation of the averaged waiting time for customers who arrive at the food truck between 12:45 PM to 1:00 PM and did not abandon the system?

```
In [5]: NEW_PM_TOTAL_WAIT_TIMES = []
        all_arrivals = 0
        balked_arrivals = 0

        for _ in range(N_DAYS):
            sim = FoodTruck(LAMBDA, SERVICE_MEAN_SEC, T0, T1)
            daily_arrivals, daily_waiting, _, balked_flags = sim.simulate_day(BALKING=True, BALKING_THRESHOLD=5)

            serviced_mask = (daily_arrivals > 105) & (daily_arrivals < 120) & (~balked_flags)
            balked_mask = (daily_arrivals > 105) & (daily_arrivals < 120) & (balked_flags)

            NEW_PM_TOTAL_WAIT_TIMES.extend(daily_waiting[serviced_mask])
            all_arrivals += np.sum(serviced_mask) + np.sum(balked_mask)
            balked_arrivals += np.sum(balked_mask)

        NEW_AVG_WAIT = np.mean(NEW_PM_TOTAL_WAIT_TIMES)
        min, sec = format_time_from_minutes(NEW_AVG_WAIT)
        print(f"Average wait time for customers arriving between 12:45 and 1:00 PM who did not balk over {N_DAYS} days: {min} minutes and {sec} seconds")
        print(f"Proportion of customers who balked between 12:45 and 1:00 PM: {balked_arrivals / all_arrivals:.2%}")
```

```
Average wait time for customers arriving between 12:45 and 1:00 PM who did not balk over 100 days: 1 minutes and 42 seconds
Proportion of customers who balked between 12:45 and 1:00 PM: 22.89%
```

f.) Re-do the previous five parts with F being an exponential distribution with expectation as 30 seconds and everything else equal.

```
In [6]: SERVICE_MEAN_SEC = 30    # new faster average service time

        AM_TOTAL_WAIT_TIMES = []
        PM_TOTAL_WAIT_TIMES = []

        for _ in range(N_DAYS):
            sim = FoodTruck(LAMBDA, SERVICE_MEAN_SEC, T0, T1)
            daily_arrivals, daily_waiting, _, _ = sim.simulate_day()

            # 11:15 - 11:30 AM arrivals
            AM_MASK = (daily_arrivals >= 15) & (daily_arrivals < 30)
            AM_WAIT_TIMES = daily_waiting[AM_MASK]
            AM_TOTAL_WAIT_TIMES.extend(AM_WAIT_TIMES)

            # 12:45 - 1:00 PM arrivals (for part b.)
            PM_MASK = (daily_arrivals >= 105) & (daily_arrivals < 120)
            PM_WAIT_TIMES = daily_waiting[PM_MASK]
            PM_TOTAL_WAIT_TIMES.extend(PM_WAIT_TIMES)

        # Part (a)
        AM_AVG_WAIT = np.mean(AM_TOTAL_WAIT_TIMES)
        min, sec = format_time_from_minutes(AM_AVG_WAIT)
        print(f"Average wait time for customers arriving between 11:15 and 11:30 AM over {N_DAYS} days: {min} minutes and {sec} seconds")
```

```python
PM_AVG_WAIT = np.mean(PM_TOTAL_WAIT_TIMES)
min, sec = format_time_from_minutes(PM_AVG_WAIT)
print(f"Average wait time for customers arriving between 12:45 and 1:00 PM over {N_DAYS} days: {min} minutes and {sec} seconds\n")

AM_over_3_min = np.sum(np.array(AM_TOTAL_WAIT_TIMES) > 3)
PM_over_3_min = np.sum(np.array(PM_TOTAL_WAIT_TIMES) > 3)
print(f"Proportion of customers waiting more than 3 minutes between 11:15 and 11:30 AM: {AM_over_3_min / len(AM_TOTAL_WAIT_TIMES):.2%}")
print(f"Proportion of customers waiting more than 3 minutes between 12:45 and 1:00 PM: {PM_over_3_min / len(PM_TOTAL_WAIT_TIMES):.2%}")

NEW_PM_TOTAL_WAIT_TIMES = []
all_arrivals = 0
balked_arrivals = 0

for _ in range(N_DAYS):
    sim = FoodTruck(LAMBDA, SERVICE_MEAN_SEC, T0, T1)
    daily_arrivals, daily_waiting, _, balked_flags = sim.simulate_day(BALKING=True, BALKING_THRESHOLD=5)

    serviced_mask = (daily_arrivals > 105) & (daily_arrivals < 120) & (~balked_flags)
    balked_mask = (daily_arrivals > 105) & (daily_arrivals < 120) & (balked_flags)

    NEW_PM_TOTAL_WAIT_TIMES.extend(daily_waiting[serviced_mask])
    all_arrivals += np.sum(serviced_mask) + np.sum(balked_mask)
    balked_arrivals += np.sum(balked_mask)

NEW_AVG_WAIT = np.mean(NEW_PM_TOTAL_WAIT_TIMES)
min, sec = format_time_from_minutes(NEW_AVG_WAIT)
print(f"Average wait time for customers arriving between 12:45 and 1:00 PM who did not balk over {N_DAYS} days: {min} minutes and {sec} seconds")
print(f"Proportion of customers who balked between 12:45 and 1:00 PM: {balked_arrivals / all_arrivals:.2%}")
```

```
Average wait time for customers arriving between 11:15 and 11:30 AM over 100 days: 3 minutes and 44 seconds
Average wait time for customers arriving between 12:45 and 1:00 PM over 100 days: 8 minutes and 17 seconds

Proportion of customers waiting more than 3 minutes between 11:15 and 11:30 AM: 48.26%
Proportion of customers waiting more than 3 minutes between 12:45 and 1:00 PM: 72.94%
Average wait time for customers arriving between 12:45 and 1:00 PM who did not balk over 100 days: 1 minutes and 14 seconds
Proportion of customers who balked between 12:45 and 1:00 PM: 16.07%
```

Problem 2

(Practice on simulating time inhomogenous Poisson processes.) Suppose that the customer arrivals from 11AM to 1PM follow a time inhomogeneous Poisson process. The arrival rate is time-varying and is linearly increasing between 11AM and 1PM. The arrival rate is 0.5 per minute at 11AM and the arrival rate is 1 per minute at 1PM. Suppose that there is one server in the system, serving customers in a first come first serve criterion. The service time requirement distribution for any customer is exponentially distributed with expectation as 35 seconds. Use 100 simulation replications for this Deliverable and provide confidence intervals with 95% of confidence level.

a) Use simulation to compute the expectation of the number of customers at time 1PM.Suppose that you are interested in knowing that on a typical Wednesday, what is the expectation of the averaged waiting time for customers who arrive at the food truck between 11:15 AM to 11:30 AM. Compute an estimate of this quantity using simulation.

b) Use simulation to compute the expectation of averaged waiting time for all those cus- tomers that arrive between 12:45 PM to 1 PM.

Note: The average waiting time is defined as for a given day, the total waiting times of all customers on that day divided by the number of customers on that day. Such average waiting time may vary from day to day and is a random variable itself. The expectation of it can be computed and approximated by taking multiple days and do a sample mean.

c.) Use simulation to compute the expectation of averaged waiting time for all those customers that arrive between 11:45 AM to 12:00 PM.

Assignment 2 – Question 2 (Solution)

Link to LLM conversation **Inhomogeneous Poisson arrivals, single-server FCFS, Exp(35s) service**
This section implements the time-change method, simulates 100 days, and reports:

- (a) Number in system at **1:00 PM (t = 120 min)**
- (b) Average wait for arrivals in **[12:45, 1:00)** → minutes `[105, 120)`
- (c) Average wait for arrivals in **[11:45, 12:00)** → minutes `[45, 60)`

  Along with **95% CIs** by normal approximation.

In [3]:
```python
import math
import random
from statistics import mean
from typing import List, Tuple, Optional

# ----------------------------
# Model parameters (minutes)
# ----------------------------
T_END = 120.0                   # 11:00 -> 1:00 mapped to t in [0, 120]
# Arrival rate: lambda(t) = 0.5 + (0.5/120) * t  customers/min
def lam(t: float) -> float:
    return 0.5 + (0.5 / 120.0) * t

# Compensator (integrated rate): Λ(t) = 0.5 t + t^2 / 480
def bigLambda(t: float) -> float:
    return 0.5 * t + (t * t) / 480.0

# Inverse of Λ: solve (1/480) t^2 + 0.5 t - y = 0  for t >= 0
# Positive root: t = [-b + sqrt(b^2 - 4 a c)] / (2 a)
# a=1/480, b=0.5, c=-y
def inv_bigLambda(y: float) -> float:
    a = 1.0 / 480.0
    b = 0.5
    c = -y
    disc = b * b - 4 * a * c  # = 0.25 + (1/120) y
    t_pos = (-b + math.sqrt(disc)) / (2 * a)
    return t_pos
```

```
# Service times i.i.d. Exp(mean = 35 seconds = 35/60 minutes)
SERVICE_MEAN_MIN = 35.0 / 60.0

def exp_sample(mean: float) -> float:
    # Inverse transform for Exp with mean
    u = random.random()
    return -mean * math.log(1.0 - u)

# ----------------------------
# Arrival generator (time-change method)
# ----------------------------
def simulate_arrivals(T: float) -> List[float]:
    """Generate arrival times in [0, T) using the time-change method."""
    arrivals = []
    y = 0.0                          # current time in Λ-space
    y_max = bigLambda(T)
    while True:
        # Jump in Λ-space ~ Exp(1)
        jump = exp_sample(1.0)       # mean-1 exponential
        y += jump
        if y >= y_max:
            break
        t = inv_bigLambda(y)
        if t < T:
            arrivals.append(t)
        else:
            break
    return arrivals

# ----------------------------
# Single-server FCFS for one day
# ----------------------------
def simulate_one_day(T: float = T_END) -> Tuple[int, Optional[float], Optional[float]]:
    """
    Returns:
      (a) number in system at t=T,
      (b) average wait among arrivals in [105,120) (minutes), or None if none,
      (c) average wait among arrivals in [45,60) (minutes), or None if none.
    """
    arrivals = simulate_arrivals(T)
    arrivals.sort()

    next_free = 0.0
    completes = []

    waits_105_120 = []
    waits_45_60 = []
    for a in arrivals:
        service = exp_sample(SERVICE_MEAN_MIN)
        start = max(a, next_free)
        wait = start - a
        finish = start + service
        next_free = finish

        completes.append(finish)

        if 105.0 <= a < 120.0:
            waits_105_120.append(wait)
        if 45.0 <= a < 60.0:
            waits_45_60.append(wait)

    # Metric (a): number in system at T
    num_arrivals_le_T = sum(1 for at in arrivals if at <= T)
    num_completes_le_T = sum(1 for ct in completes if ct <= T)
    number_in_system_at_T = num_arrivals_le_T - num_completes_le_T

    avg_wait_105_120 = (mean(waits_105_120) if waits_105_120 else None)
    avg_wait_45_60 = (mean(waits_45_60) if waits_45_60 else None)

    return number_in_system_at_T, avg_wait_105_120, avg_wait_45_60

# ----------------------------
# Replications + 95% CI
# ----------------------------
def ci_mean_95(samples):
    """Return (mean, lo, hi) using normal approx: mean ± 1.96 * s / sqrt(n)."""
    n = len(samples)
    m = mean(samples) if n else float('nan')
    if n <= 1:
        return (m, m, m)
    s2 = sum((x - m) ** 2 for x in samples) / (n - 1)
    s = s2 ** 0.5
    half = 1.96 * s / (n ** 0.5)
    return (m, m - half, m + half)

def run(days: int = 100, seed: Optional[int] = 123):
    if seed is not None:
        random.seed(seed)

    a_vals = []  # number in system at 120
    b_vals = []  # avg wait [105,120)
    c_vals = []  # avg wait [45,60)

    for _ in range(days):
        a, b, c = simulate_one_day(T_END)
        a_vals.append(float(a))
        if b is not None: b_vals.append(b)
        if c is not None: c_vals.append(c)

    a_mean, a_lo, a_hi = ci_mean_95(a_vals)
    if b_vals:
        b_mean, b_lo, b_hi = ci_mean_95(b_vals)
    else:
        b_mean = b_lo = b_hi = float('nan')
    if c_vals:
        c_mean, c_lo, c_hi = ci_mean_95(c_vals)
    else:
        c_mean = c_lo = c_hi = float('nan')

    results = {
        "days": days,
        "count_days_b": len(b_vals),
        "count_days_c": len(c_vals),
        "metric_a_mean": a_mean,
        "metric_a_CI": (a_lo, a_hi),
        "metric_b_mean_min": b_mean,
        "metric_b_CI_min": (b_lo, b_hi),
        "metric_b_mean_sec": 60*b_mean if math.isfinite(b_mean) else float('nan'),
        "metric_b_CI_sec": (60*b_lo, 60*b_hi) if math.isfinite(b_lo) and math.isfinite(b_hi) else (float('nan'), float('nan')),
        "metric_c_mean_min": c_mean,
        "metric_c_CI_min": (c_lo, c_hi),
        "metric_c_mean_sec": 60*c_mean if math.isfinite(c_mean) else float('nan'),
        "metric_c_CI_sec": (60*c_lo, 60*c_hi) if math.isfinite(c_lo) and math.isfinite(c_hi) else (float('nan'), float('nan')),
    }
    return results, a_vals, b_vals, c_vals

# Quick sanity: expected number of arrivals in 120 minutes is Λ(120).
EXPECTED_ARRIVALS = bigLambda(120.0)
EXPECTED_ARRIVALS
```

Out[3]:
```
90.0
```

In [4]:
```
# ---- Execute 100-day simulation ----
results, a_vals, b_vals, c_vals = run(days=100, seed=123)

print("=== Simulation Results (100 days) ===")
print("(a) Number in system at t=120 min:")
print(f"    mean = {results['metric_a_mean']:.3f},  95% CI = [{results['metric_a_CI'][0]:.3f}, {results['metric_a_CI'][1]:.3f}]")

print("\n(b) Avg wait for arrivals in [105,120) minutes:")
print(f"    mean = {results['metric_b_mean_min']:.4f} min  ({results['metric_b_mean_sec']:.2f} s)")
print(f"    95% CI = [{results['metric_b_CI_min'][0]:.4f}, {results['metric_b_CI_min'][1]:.4f}] min  "
      f"([{results['metric_b_CI_sec'][0]:.2f}, {results['metric_b_CI_sec'][1]:.2f}] s)")
print(f"    computed over {results['count_days_b']} days with ≥1 arrival in that window.")
print("\n(c) Avg wait for arrivals in [45,60) minutes:")
print(f"    mean = {results['metric_c_mean_min']:.4f} min  ({results['metric_c_mean_sec']:.2f} s)")
print(f"    95% CI = [{results['metric_c_CI_min'][0]:.4f}, {results['metric_c_CI_min'][1]:.4f}] min  "
      f"([{results['metric_c_CI_sec'][0]:.2f}, {results['metric_c_CI_sec'][1]:.2f}] s)")
print(f"    computed over {results['count_days_c']} days with ≥1 arrival in that window.")

print("\nSanity check: Expected arrivals in 120 min by Λ(120) = 90.0")
print(f"Λ(120) from code = {EXPECTED_ARRIVALS:.1f} (should be 90.0)")
```

```
=== Simulation Results (100 days) ===
(a) Number in system at t=120 min:
    mean = 1.230,  95% CI = [0.914, 1.546]

(b) Avg wait for arrivals in [105,120) minutes:
    mean = 0.5823 min  (34.94 s)
    95% CI = [0.4445, 0.7200] min  ([26.67, 43.20] s)
    computed over 100 days with ≥1 arrival in that window.

(c) Avg wait for arrivals in [45,60) minutes:
    mean = 0.3332 min  (19.99 s)
    95% CI = [0.2395, 0.4269] min  ([14.37, 25.61] s)
    computed over 100 days with ≥1 arrival in that window.

Sanity check: Expected arrivals in 120 min by Λ(120) = 90.0
Λ(120) from code = 90.0 (should be 90.0)
```