MA424 Summative Project

Assessment Candidate Numbers: 70320, 68729, 61115

1 The first time Foodys is under-resourced

1.1

As specified in the question, we use Discrete Event Simulation (DES) to model Foodys' scooter fleet operations. Here, the system consists of two queues (one for check-ups, one for repairs), each with one server (mechanic 1 and 2). The arrival process has two components: a deterministic part (mandatory check-ups every $c = 2$ consecutive days of work) and a stochastic part (breakdowns following an exponential distribution with rate $\mu = 0.25$ per day). In our setup, we have $N = 20$ scooters, $n = 10$ working at any time, and 10 in reserve. Thus, an arrival is when scooter breaks down (case 1) or needs a check-up (case 2), it then goes to the relevant mechanic and a scooter in the parking replaces it. On the other hand, a departure is when mechanic 1 finishes servicing a scooter which needed a check-up (case 3) or when mechanic 2 finishes repairing a broken down scooter (case 4), in both cases they are then put it back in the parking. We want to simulate until T, or the first time Foodys becomes under-resourced (i.e. when reserves run out).

Variables:

1. time t
2. system state $SS = (Q_1, Q_2, W, P)$ where:
   - Q_1 = scooters in the check-up queue system (including one being served)
   - Q_2 = scooters in the repair queue system (including one being served)
   - W = set of working scooters at time t
   - P = set of scooters in the parking lot at time t
3. counter variables:
   - c_n = vector of scheduled check-up times for each scooter $n = 1, \ldots, N$ (∞ if not working and $t + 2$ if working)
   - b_n = vector of scheduled breakdown times for each scooter $n = 1, \ldots, N$ (∞ if not working and $t + Y$ if working, with $Y = -\frac{1}{\mu}\log(U)$)

Event List: $EL = (t_{A,1}, t_{A,2}, t_1, t_2)$ where:

- $t_{A,1} = \min(c_n)$ = time of next check-up arrival (earliest scheduled check-up)
- $t_{A,2} = \min(b_n)$ = time of next breakdown arrival (earliest breakdown)
- t_1 = time of service completion at mechanic 1 (check-ups), ∞ if idle
- t_2 = time of service completion at mechanic 2 (repairs), ∞ if idle

Output Variables: T = the first time Foodys becomes under-resourced (i.e., when the parking lot P is empty and a working scooter needs to be replaced)

Explanation of New Variables: Compared to a standard single-server queue, we introduce several new variables. First, we track individual scooters rather than anonymous customers using sets W (working) and P (parking), because each scooter has its own history and scheduled events. Second, we maintain separate queues Q_1 and Q_2 for the two types of service. Third, we use vectors c_n and b_n to track when each individual scooter is due for a check-up or will break down. These are necessary because the arrival process is not a simple Poisson process but depends on each scooter's working history. The event list contains four event types (two arrivals, two departures) instead of the usual two.

Algorithm Pseudocode:

Initialize

1. Set $t = 0$.
2. Set SS: $W = \{1, 2, \ldots, n\}$, $P = \{n + 1, \ldots, N\}$, $Q_1 = \emptyset$, $Q_2 = \emptyset$.
3. For $i = 1, \ldots, n$: generate U and set $b_n[i] = t - \frac{1}{\mu}\log(U)$. Set $c_n[i] = \infty$.
4. For $i = n + 1, \ldots, N$: set $b_n[i] = \infty$, $c_n[i] = \infty$.
5. Set $t_1 = t_2 = \infty$.

Then depending on the event list $EL = (\min(c_n), \min(b_n), t_1, t_2)$ we have one of the following cases:

Case 1: $\min(b_n) = \min\{\min(c_n), \min(b_n), t_1, t_2\}$ **(Breakdown - arrival in Q_2)**

1. Set $t = \min(b_n)$. Let $i = \arg\min(b_n)$ (index of the scooter that broke down).
2. Remove i from W. Add i to Q_2.
3. Set $b_n[i] = \infty$, $c_n[i] = \infty$.
4. If $P = \emptyset$: set $T = t$ and STOP (Foodys is under-resourced).
5. Else: let j = last element of P. Remove j from P. Add j to W.
   Generate U, set $Y = -\frac{1}{\mu}\log(U)$ and set $b_n[j] = t + Y$, $c_n[j] = t + c$.
6. If $|Q_2| = 1$: generate U, set $Y = -\frac{1}{\mu_2}\log(U)$ and set $t_2 = t + Y$.

Case 2: $\min(c_n) = \min\{\min(c_n), \min(b_n), t_1, t_2\}$ **(Check-up - arrival in Q_1)**

1. Set $t = \min(c_n)$. Let $i = \arg\min(c_n)$ (index of the scooter due for check-up).

2. Remove i from W. Add i to Q_1.
3. Set $b_n[i] = \infty$, $c_n[i] = \infty$.
4. If $P = \emptyset$: set $T = t$ and STOP (Foodys is under-resourced).
5. Else: let j = last element of P. Remove j from P. Add j to W.
   Generate U, set $Y = -\frac{1}{\mu}\log(U)$ and set $b_n[j] = t + Y$, $c_n[j] = t + c$.
6. If $|Q_1| = 1$: generate U, set $Y = -\frac{1}{\mu_1}\log(U)$ and set $t_1 = t + Y$.

Case 3: $t_1 = \min\{\min(c_n), \min(b_n), t_1, t_2\}$ **(Departure from mechanic 1)**

1. Set $t = t_1$. Let i = first element of Q_1 (FIFO).
2. Remove i from Q_1. Add i to P.
3. If $Q_1 = \emptyset$: set $t_1 = \infty$.
4. Else: generate U, set $Y = -\frac{1}{\mu_1}\log(U)$ and set $t_1 = t + Y$.

Case 4: $t_2 = \min\{\min(c_n), \min(b_n), t_1, t_2\}$ **(Departure from mechanic 2)**

1. Set $t = t_2$. Let i = first element of Q_2 (FIFO).
2. Remove i from Q_2. Add i to P.
3. If $Q_2 = \emptyset$: set $t_2 = \infty$.
4. Else: generate U, set $Y = -\frac{1}{\mu_2}\log(U)$ and set $t_2 = t + Y$.

R Implementation: The R script implements this DES model (using Ex.3 Seminar 8 as a basis). The outer loop runs $K = 500$ independent simulations with the same initial conditions. Within each simulation, the inner while-loop processes events one at a time. First, at each iteration, we compute the event list by finding the minimum of the earliest checkup time, earliest breakdown time, and the two service completion times. The code then branches into one of four cases shown above in the pseudocode. During the iteration, vectors W, P store scooter indices and Q_1, Q_2 store the two queues in order (FIFO). Further, vectors b_n and c_n store scheduled breakdowns and check-ups for all N scooters and are updated each time one of these events happens (with ∞ values indicating no scheduled event). The simulation terminates when the parking lot is empty and a replacement is needed. We then record each simulation's stopping time T and compute the mean across the K iterations. Finally, we note that the script uses only `runif()` for random number generation and begins with `set.seed(1)` for reproducibility.

Final Results and Interpretation: Our final estimate of $E[T]$ is computed as the sample mean of the 500 values of T. With the given parameters ($N = 20$, $n = 10$, $c = 2$, $\mu = 0.25$, $\mu_1 = 1.5$, $\mu_2 = 2$), we obtain $\mathbf{E[T] \approx 6.4236}$ **days** with the standard deviation of the estimator being 0.0985. To interpret this estimate, we note that we have 10 working scooters with a breakdown rate of $\mu = 0.25$ per day resulting in a combined breakdown rate of 2.5 scooters per day. Further, mechanic 2 repairs $\mu_2 = 2$ scooters per day which is slightly lower than the combined breakdown rate, thus, we have a slower effect on T. On the other hand, once the first 10 scooters have broken down (which happens in 4 days on average given the 10 scooters and the combined breakdown rate of 2.5 per day), all replacement scooters will have a scheduled check-up after 2 days. Since a scooter's probability of surviving until check-up time, where $X \sim \text{Exp}(\mu)$, is $P[X > 2] = e^{-0.25 \cdot 2} \approx 0.61$ or 61%, then most scooters will go to check-up where mechanic 1 can only service $\mu_1 = 1.5$ scooters per day. This will lead to an imbalance where the check-up arrival rate will ultimately exceed mechanic 1's servicing capability and lead to an under-resourced situation. Thus, our 6.42 days value reflects the fact that the bottleneck at the check-up queue takes effect fast after all first 10 scooters have broken down.

1.2

To find an estimate of $E[T]$ where we are 95% confident that the estimate is within 0.3 days of its true value, we first need to find the acceptable value for the standard deviation of the estimator, which we call d. From the Central Limit Theorem (CLT), we know that $P[-z_{\alpha/2} \cdot d < \bar{T} - E[T] < z_{\alpha/2} \cdot d] = 1 - \alpha$ where $d = S/\sqrt{k}$ is the standard deviation of the estimator. Thus, since we need 95% confidence we need $z_{0.025} \cdot d = 0.3$ resulting in $d = \frac{0.3}{1.96} \approx 0.153$. We then must simulate values of T, and we simulate at least 100 iterations before stopping given that the CLT must hold. Afterwards, we only stop once $S/\sqrt{k} < d$, i.e. once the standard deviation of the estimator $\bar{T}$ is strictly lower than our acceptable value d.

To find how many iterations are needed to reach a standard deviation of d we add a new condition within our previous R script's loop. We start by defining $d = 0.3/1.96$ and we define function **gen_sample_mean_var** from Ex.3 seminar 8. Once in our while loop, we initialise the mean and variance of our iteration when $k = 1$ and then, for all other iterations, we update them using the function defined prior. Finally, we define an if statement which prints the iterations number, variance and standard deviation once we have done at least 100 iterations ($k \geq 100$), and the standard deviation is strictly smaller than d. Within the if statement, we also break our while loop, thus, exiting the loop.

Our results show that we reached $S/\sqrt{k} \approx 2.1402/\sqrt{196} \approx 0.1529 < d$ in $k = 196$ iterations with an estimator $E[T] = 6.24$ and a sample variance of 4.5802.

1.3

We give an interval around $\bar{T} = 6.24$ (from Q1.2) which we are 90% confident that the true value of $E[T]$ lies (we call this true value θ). We start by assuming that, by the CLT, as k (or the number of iterations) grows larger the distribution $\frac{(\bar{T}-\theta)}{S/\sqrt{k}}$ is approximately $N(0, 1)$. In the previous normalisation, S is the sample standard deviation, thus, $S/\sqrt{k}$ is the standard deviation of the estimator. Now, we can simply calculate the interval as the product between the z-score for our chosen confidence level (here $z_{\alpha/2} = z_{0.05} = 1.645$) and the standard error $\sigma/\sqrt{k}$ which we estimate with $S/\sqrt{k} \approx 2.1402/\sqrt{196} \approx 0.1529$ (this substitution

remains valid as per Slutsky's theorem). Thus, we get: $\bar{T} \pm 1.645 \cdot 0.1529 = 6.24 \pm 0.2515$ which is the interval $[5.989, 6.492]$. To conclude, this means that if we repeat the procedure from Q1.2 many times, around 90% of the intervals constructed will contain the true value θ.

We now show that this is true analytically. Assuming that our target confidence is α, then as per the CLT, we know $P[Z > z_\alpha] = \alpha$ where Z is a z-score and z_α is our chosen confidence level z-score. We then have:

$$\begin{aligned}
P\left[-z_{\alpha/2} < Z < z_{\alpha/2}\right] &\approx 1-\alpha && \text{for } Z \sim N(0,1) \\
P\left[-z_{\alpha/2} < \frac{\theta - \bar{T}}{S/\sqrt{k}} < z_{\alpha/2}\right] &\approx 1-\alpha && \text{substitute } Z = \frac{\bar{T}-\theta}{S/\sqrt{k}} \text{ \& use symmetry} \\
P\left[-z_{\alpha/2}(S/\sqrt{k}) < \theta - \bar{T} < z_{\alpha/2}(S/\sqrt{k})\right] &\approx 1-\alpha && \text{multiply by } S/\sqrt{k} \\
P\left[\bar{T} - z_{\alpha/2}(S/\sqrt{k}) < \theta < \bar{T} + z_{\alpha/2}(S/\sqrt{k})\right] &\approx 1-\alpha && \text{add } \bar{T}
\end{aligned}$$

Thus, setting $\alpha = 0.1$ we get $P[\bar{T} - z_{0.05} \cdot S/\sqrt{k} < \theta < \bar{T} + z_{0.05} \cdot S/\sqrt{k}] \approx 1 - 0.1 = 0.9$ which is equivalent to $\bar{T} \pm z_{0.05} \cdot S/\sqrt{k}$, hence, proving analytically that our earlier calculation is, indeed, a 90% confidence interval.

1.4

In order to improve the checkup/repair efficiency at Foodys, we recommend changing from the current dedicated server system to a pooled resource system. Since the problem description states that both mechanics are qualified to handle both checkups and repairs, they should be flexible in their assignments. This could be implemented by having a single central queue for all incoming scooters, rather than separate queues for each task.

The current system is inefficient because it allows for a state where one mechanic is idle while the other has a queue of scooters waiting. For example, if no checkups are required, Mechanic 1 sits idle even if Mechanic 2 is overwhelmed with repairs. This "wasted capacity" effectively lowers the overall service rate of the system. Indeed, a fundamental result in queueing theory is that pooled servers sharing a common queue achieve lower expected waiting times than equivalent separate single-server queues.

By pooling the resources, we eliminate this inefficiency, as a mechanic never sits idle as long as there is any work to be done in the shop, indeed, combined service rate becomes $\mu_1 + \mu_2 = 3.5$ per day. The variable T is determined by the ability of the shop to repair scooters fast enough to keep the parking lot from emptying. By eliminating idle times, the average time a scooter spends in the shop decreases. This increases the throughput rate at which scooters are returned to the parking lot, thereby maintaining a higher buffer level of spares and increasing the expected time $E[T]$ until the system becomes under-resourced.

2 Reduce the variance of the estimator

To reduce the variance of our estimator for $E[T]$, we propose using the sample mean of the breakdown times for the initial n working scooters as a control variate.

Let $B_1, B_2, \ldots, B_n$ be the randomly generated times until the first breakdown for the $n = 10$ initial working scooters. We define this control variate Y as:

$$Y = \frac{1}{n}\sum_{i=1}^{n} B_i$$

According to the method of control variates described in Lecture 10, a valid control variate Y must satisfy two main conditions:

1. **Have a Known Expected Value:** We must know $\mu_Y = E[Y]$. Since the initial breakdown times B_i are generated from an exponential distribution with rate $\mu = 0.25$, we know theoretically that $E[B_i] = \frac{1}{\mu} = 4$. Therefore, the expected value of their sample mean is also exactly known:

$$E[Y] = E[\frac{1}{n}\sum_{i=1}^{n} B_i] = \frac{1}{n}\sum_{i=1}^{n} E[B_i] = \frac{1}{n} \cdot n \cdot 4 = 4$$

2. **Strong Correlation to Output:** The variable Y must be strongly correlated with the output T. In this system, the time T is heavily dependent on how long the initial fleet survives. If the initial scooters happen to last longer than average (i.e., a high Y value), the system draws from the parking lot more slowly, leading to a larger T. Thus, Y and T are strongly positively correlated.

Now, we create a linear combination of T, Y with a constant c and we show that it remains an unbiased estimator of $E[T] = \theta$, indeed, we have the control estimator $T_{control} = T + c(Y - \mu_Y)$:

$$E[T_{control}] = E[T + c(Y - \mu_Y)] = E[T] + cE[Y] - c\mu_Y = E[T] = \theta$$

As per the derivation shown in lecture 10, to reduce the variance of $E[T]$ we much choose constant c such that we minimise the variance of $T_{control}$, this results in optimal constant $c^* = -\frac{\text{Cov}[T,Y]}{\text{Var}[Y]}$. To understand why correlation between T, Y directly influences the variance, we plug-in c^* in the original variance:

$$\text{Var}[T + c(Y - \mu_Y)] = \text{Var}[T] + c^2\text{Var}[Y] + 2c\text{Cov}[T,Y] = \text{Var}[T] + \frac{\text{Cov}[T,Y]^2}{\text{Var}[Y]^2}\text{Var}[Y] - 2\frac{\text{Cov}[T,Y]^2}{\text{Var}[Y]} = \text{Var}[T] - \frac{\text{Cov}[T,Y]^2}{\text{Var}[Y]}$$

By dividing with $\text{Var}[T]$ we then get the below:

$$\frac{\text{Var}[T + c(Y - \mu_Y)]}{\text{Var}[T]} = 1 - \frac{\text{Cov}[T, Y]^2}{\text{Var}[Y]\text{Var}[T]} = 1 - \rho_{T,Y}^2$$

Thus, we have shown that by choosing the unbiased estimator $T_{control}$ we are able to reduce the variance of $E[T]$ by $100 \cdot \rho_{T,Y}^2$ percent. Thus, since the variance of the controlled estimator is $\text{Var}[T](1 - \rho_{T,Y}^2)$ higher correlation (either positive or negative) leads to greater variance reduction. This justifies our choice of Y being the average of the first 10 breakdown times as it is highly positively correlated with T.

3 Correct the distribution of the mechanics

3.1

To simulate the service times for the two mechanics, we analyse the properties of their given distributions and select the most appropriate method based on the principles of generating random variables. For Mechanic 2, the Cumulative Distribution Function (CDF) is explicitly given as:

$$F(x) = 1 - e^{-3x^2}, \quad x > 0$$

According to Proposition 2.2 in the lecture notes, the Inverse Transform Method is the most efficient approach when the CDF $F(x)$ is known and algebraically invertible. Since $F(x)$ consists of standard exponential and polynomial terms, we can directly solve for x in terms of a uniform random variable.
Let U be a random variable distributed uniformly on $(0, 1)$, i.e., $U \sim \text{Uniform}(0, 1)$. We set $u = F(x)$ and solve for x:

$$\begin{aligned} u &= 1 - e^{-3x^2} \\ e^{-3x^2} &= 1 - u \\ -3x^2 &= \ln(1 - u) \\ x^2 &= -\frac{1}{3}\ln(1 - u) \\ x &= \sqrt{-\frac{1}{3}\ln(1 - u)} \end{aligned}$$

Since $U \sim \text{Uniform}(0, 1)$, the variable $(1 - U)$ is also uniformly distributed on $(0, 1)$. For computational simplicity, we can replace $(1 - U)$ with U. Thus, our generator algorithm is:

1. Generate $U \sim \text{Uniform}(0, 1)$.
2. Return $X = \sqrt{\frac{-\ln(U)}{3}}$.

Our R script follows the above algorithm (using seminar 2 R codes as a basis).
Now for Mechanic 1, we are given the Probability Density Function (PDF):

$$f(x) = \frac{1}{2}(1 + x)e^{-x}, \quad x > 0$$

To use the Inverse Transform Method, we would first need the CDF. Integrating $f(x)$ yields:

$$F(x) = \int_0^x \frac{1}{2}(1 + t)e^{-t}dt = 1 - \frac{1}{2}(2 + x)e^{-x}$$

Solving $u = F(x)$ for x is analytically impossible because x appears both as a linear term and in the exponent. Therefore, as described in Section 2.2.2 of the lecture notes, we must use the Acceptance-Rejection Method.
A critical requirement for the Acceptance-Rejection method is that the ratio $f(x)/g(x)$ must be bounded by a finite constant c. This implies that the envelope density $g(x)$ must decay more slowly than the target density $f(x)$.
Our target density $f(x)$ is dominated by the term e^{-x}. If we were to choose a candidate distribution with the same rate (e.g., $g(x) \propto e^{-x}$), the polynomial term $(1 + x)$ in the numerator would cause the ratio $f(x)/g(x)$ to diverge to infinity as $x \to \infty$.
To ensure the ratio remains bounded, we select an Exponential distribution with a rate $\lambda < 1$, which decays slower than the target. We choose $\lambda = 0.5$:

$$g(x) = 0.5e^{-0.5x}, \quad x > 0$$

Since this is function integrates to 1 and g(x) is strictly positive when f(x) is strictly positive, it satisfies the rules of an envelope density.
We can easily simulate from this distribution using the Inverse Transform Method ($Y = -2\ln(U)$) as we have often done when simulating exponential distributions in the lectures ($-\frac{1}{\mu} \cdot \log(U)$ where μ is the rate of decay).
Then we need to find a constant c such that $f(x) \leq c \cdot g(x)$ for all x. This is equivalent to finding the maximum of the ratio $h(x) = f(x)/g(x)$, doing this also ensures that we find a good value of c given that the acceptance efficiency is $1/c$.

$$h(x) = \frac{0.5(1+x)e^{-x}}{0.5e^{-0.5x}} = (1+x)e^{-0.5x}$$

To maximize $h(x)$, we take the derivative with respect to x:

$$h'(x) = 1 \cdot e^{-0.5x} + (1+x)(-0.5)e^{-0.5x} = e^{-0.5x}(1 - 0.5 - 0.5x) = e^{-0.5x}(0.5 - 0.5x)$$

Setting $h'(x) = 0$ gives the critical point at $x = 1$. The maximum value is:

$$c = h(1) = (1+1)e^{-0.5(1)} = 2e^{-0.5} \approx 1.213$$

Based on the Rejection Method theorem, our algorithm is:

1. **Generate Candidate:** Generate $Y \sim \text{Exp}(0.5)$ by setting $Y = -2\ln(U_1)$ where $U_1 \sim \text{Uniform}(0, 1)$.
2. **Generate Test Variable:** Generate $U_2 \sim \text{Uniform}(0, 1)$.
3. **Accept/Reject:** Check if $U_2 < \frac{f(Y)}{c \cdot g(Y)}$.
    - Substituting our simplified ratio: Check if $U_2 < \frac{(1+Y)e^{-0.5Y}}{1.213}$.
    - If true, return $X = Y$. Else, return to Step 1.

Again, our R script follows the above algorithm (using seminar 2 R codes as a basis).

3.2

To verify the correctness of our algorithms, we generated $N = 10,000$ independent and identically distributed (i.i.d.) random values using the functions defined in Part 1. We plotted the histograms of these simulated values and overlaid the theoretical probability density functions (PDFs).

A standard frequency histogram displays the count of observations in each bin, whereas a probability density function (PDF) $f(x)$ integrates to 1. To visually compare the two on the same scale, we must multiply the theoretical density by a scaling constant C. The relationship is derived as follows:

- The total area of the histogram bars is: Total Area = Total Samples$(N) \times$ Bin Width(W).
- The total area under the PDF curve is 1.
- Therefore, to match the scale of the frequency histogram, we use the constant:

$$C = N \times W$$

In our plots, the theoretical curve $y = C \cdot f(x)$ is plotted over the histogram.

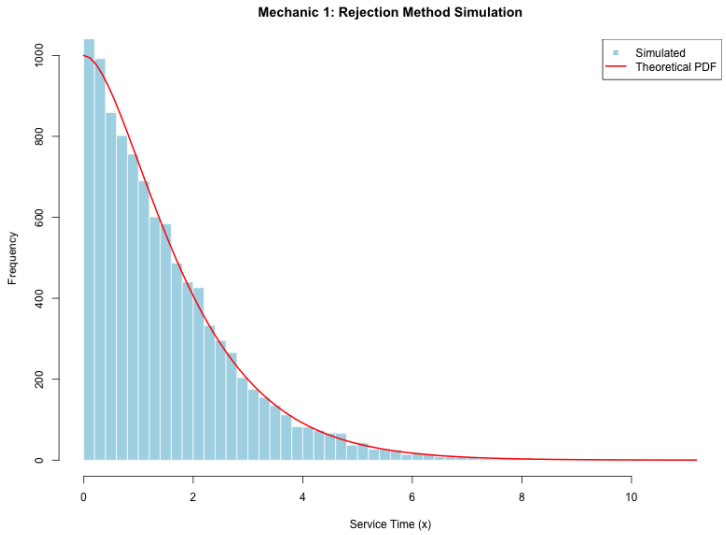

Figure 1: Histogram of 10,000 simulated service times for Mechanic 1 (Rejection Method) overlaid with the scaled theoretical density $f(x) = \frac{1}{2}(1+x)e^{-x}$.

Figure 1 shows the results for the Acceptance-Rejection method for mechanic 1. The histogram (light blue bars) closely follows the red theoretical curve. The distribution exhibits the expected behaviour: starting with a non-zero probability at $x = 0$, rising slightly, and then following an exponential decay. The close alignment confirms that our choice of the envelope $g(x) \sim \text{Exp}(0.5)$ and the rejection constant $c \approx 1.213$ were correct.

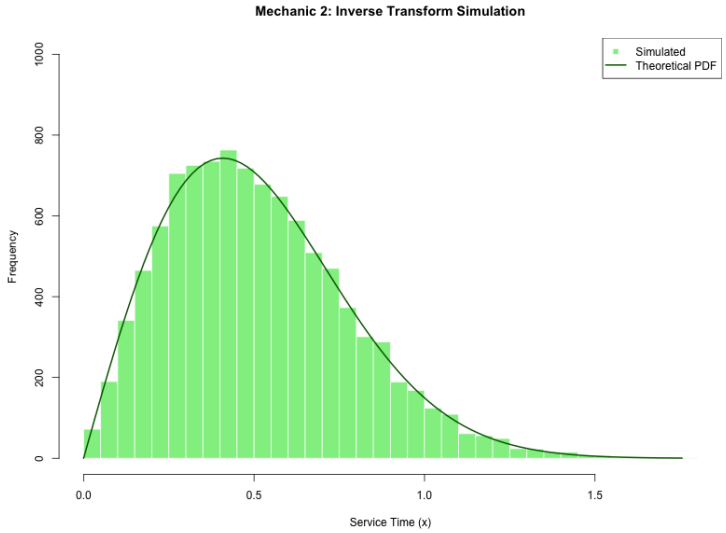

Figure 2: Histogram of 10,000 simulated service times for Mechanic 2 (Inverse Transform Method) overlaid with the scaled theoretical density $f(x) = 6xe^{-3x^2}$ (derived from F(x) using the chain rule).

Figure 2 shows the results for the Inverse Transform method for mechanic 2. The histogram aligns perfectly with the dark green theoretical curve derived from the CDF $F(x) = 1 - e^{-3x^2}$. The data captures the distinct shape of this distribution, which starts at 0, peaks around $x \approx 0.4$, and decays rapidly. This confirms that the inverse CDF formula $x = \sqrt{-\frac{1}{3}\ln(U)}$ was derived and implemented correctly.
In both cases, the simulated data is almost indistinguishable from the theoretical distributions, verifying that the random variate generation algorithms for both mechanics are accurate.

4 CEO's visit to r cities

4.1

In this question, we use the Metropolis-Hastings (MH) algorithm since our state space consists of permutations with no natural joint distribution for Gibbs sampling. Taking inspiration from the knapsack problem (Example 4, Lecture 9), MH allows us to explore the 10! possible itineraries efficiently by accepting/rejecting proposed permutation swaps.

Next, our chosen stationary distribution $s(x)$ must peak at high-valued permutations of the CEO's itinerary i.e. during the simulation our chain would spend most of its time at high-valued itineraries and rarely moves to ones with lower value. Here, we look at the knapsack problem (example 4 lecture 9) because we believe that the same $s(x)$ function is appropriate here. Indeed, we choose $s(x) \propto e^{\beta V(x)}$ where β is a tunable parameter and $V(x)$ is our reward function, because this exponential function assigns more probability to high valued outcomes. This aligns exactly to our current objective. Further, we note that using MCMC is needed given that the normalising constant for this distribution $S = \Sigma_{x \in C} e^{\beta V(x)}$ has 10! terms so cannot be computed. To determine the acceptance probability, we note that the symmetry of our proposal chain enables us to simplify the acceptance probability to $a(x, y) = \min(1, e^{\beta(V(y)-V(x))})$. Further, we illustrate how the β parameter adjusts the likeliness accepting or rejecting the proposal chain we generated. Here, we assume that the difference of value between the proposal chain and current chain is -1. This implies that the ratio of their probabilities equates to: $s(y)/s(x) = e^{\beta \cdot V(y)}/e^{\beta \cdot V(x)} = e^{\beta \cdot (V(y)-V(x))} = e^{-\beta}$). Thus, acceptance probabilities range from 99% ($\beta = 0.01$), 90% ($\beta = 0.1$), to 37% ($\beta = 1$), showing β controls exploration. To conclude, while lower β values enable our algorithm to "explore" the state space more freely (as illustrated prior), higher values will tend "exploit" (or focus on) higher value proposal chains only. This implies that the markov process will display three transition states when changing betas: 1. exploration, 2. balanced exploration-exploitation, 3. exploitation. With this in mind, we decide to simulate $\beta \in \{0, 1, 2, ..., 20\}$ in steps of 0.1 and plot their averages, this will help us choose the three relevant beta values:

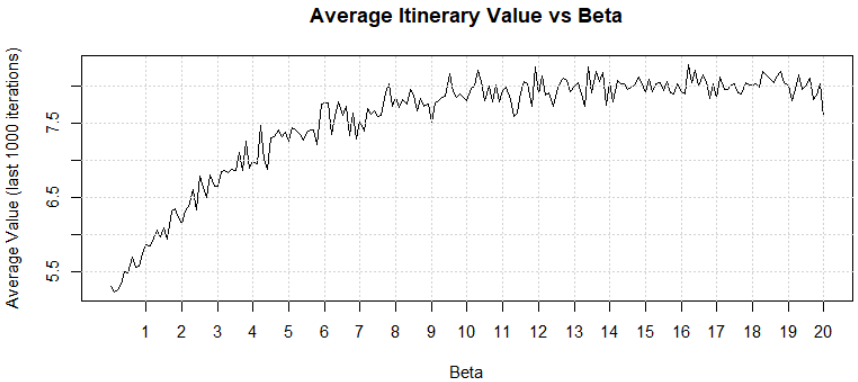

Figure 3: Averages of the 1000 last simulated itinerary values for $\beta \in \{0, 1, ..., 20\}$ in steps of 0.1

As seen in Figure 3, the averages of each distribution simulated clearly show three distinct behaviours. First, at low betas (<0.5), the chain explores the state space freely resulting in a low average. Second, betas between 0.5 and 10 increasingly focus on exploitation, explaining why averages in this range continue increasing. Finally, betas above 10 focus on exploitation, thus, getting "stuck" in high value states and rarely explore the state space, these states can sometimes be good but they can also be sub-optimal. Within these three ranges, we arbitrarily pick representative values of $\beta \in \{0.1, 3, 12\}$. Lastly, when simulating each beta value, we simulate 10,000 iterations of the MH algorithm as this typically enables us to converge towards the stationary distribution and forget the initial state.

R Implementation: We now explain the R script we created (using Ex.3 Seminar 9 as a basis) to simulate high value itineraries for Foodys' CEO visiting 10 different cities ($r = 10$). We define the fixed variables: $N = 10,000$, $L = 1,000$ (plotted iterations), $r = 10$, $beta\_values$ (vector βs). We then generate the reward matrix v defining the value of going from city i (rows) to j (columns). We initialise it with zeros $v \in \mathbb{R}^{r+1 \times r+1}$. Then, using Uniform$(0, 1)$ random variables, we generate $v(0, j)$ for $j = 1, ..., 10$ (rewards for leaving the starting city). After, we repeat this for all pairs $v(i, j)$ where $i \neq j$. This ensures that the value of returning at city 0 (the starting city) is always zero. We then define a reward function V taking the current itinerary $x = (x_1, x_2, ..., x_{10})$ and the reward matrix v as arguments. This function calculates the total value by adding the value of the 0 to x_1 path ($v(0, x_1)$) and adding the value of each individual itinerary $v(x_i, x_{i+1})$ to our total value (this includes 10 cities since we use index $i + 1$) so: $v(0, x_1) + \sum_{i=1}^{r-1} v(x_i, x_{i+1})$.

As a second step, we create the main loops for the algorithm. In the outer loop, we initialise our beta from the $beta\_values$ vector and our first permutation of cities as $1, 2, ..., 10$ (arbitrarily chosen for simplicity). We also initialise vector X which stores the total values of each simulated permutation. We then simulate the remaining N-1 steps. We do this by initialising Y to be equal to the current itinerary X and (inspired from the lecture 9 knapsack) swapping two indices i and j (chosen uniformly at random from from $1, 2, ..., r$) within Y. This always creates a new valid permutation Y (no need to check for feasibility). Also, we decided not to check for repeating permutations (could be done using rejection sampling). After creating Y, we compute the value of the current itinerary $X_{current}$ and the proposed itinerary Y using function V. Naturally, the next step is to calculate the acceptance probability $min(1, e^{\beta(V(Y)-V(X))})$ which equates to stating that if the proposed itinerary has a lower value than the current one (i.e. $V(Y) < V(X)$ or $e^{\beta(V(Y)-V(X))} < 1$) then we accept our proposed itinerary with $P[U < e^{\beta(V(Y)-V(X))}] = e^{\beta(V(Y)-V(X))}$ while if the opposite is true, we always accept the proposed chain $P[U < 1] = 1$. Once we have accepted/rejected Y, and updated $X\_current$ accordingly, we then store the new itinerary's value in the X vector and repeat the inner loop for all chosen β values. After the inner for loop finishes, the next step in the outer loop is to print the average of 1000 last values for the chosen betas $\{0, 1, 2, ..., 20\}$ and store this in the $Averages$ vector (initialised prior).

We then make our choice of three betas to display different MC behaviours. Afterwards, we reinitialise seed(1), re-define the $beta\_values$ vector with $\beta \in \{0.1, 3, 12\}$, and repeat our previous loop for these new values of beta. Finally, we create a plot area and plot the averages graph from our original beta simulation and then add three histograms (for the 1000 last values) after the end of our $\beta \in \{0.1, 3, 12\}$ loop. We choose the 1000 last values because this is when our chosen itineraries have converged to the stationary distribution and have forgotten the initial state. Thus, we plot from index $N - L + 1$ to N of our X vector.

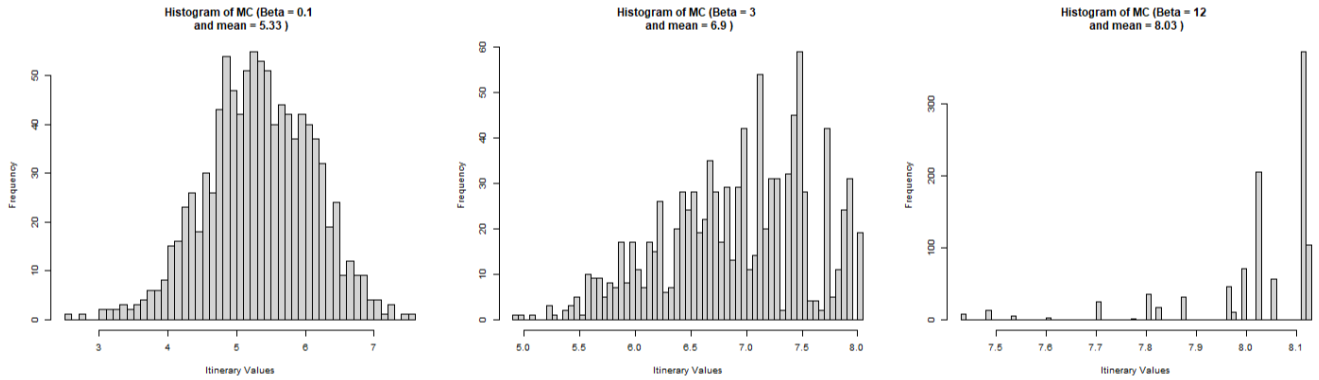

Figure 4: Histogram of the 1000 last simulated itinerary values for $\beta \in \{0.1, 3, 12\}$ with averages of 5.33, 6.90, and 8.03, respectively

To conclude, the histograms in Figure 4 demonstrate three distinct MC behaviours. We had $\beta = 0.1$ the exploration regime, $\beta = 3$ the transition regime, and $\beta = 12$ the exploitation regime. Looking at $\beta = 0.1$, the histogram displays a wide, roughly symmetric distribution centred around its mean of ≈ 5.3. The last 1000 itineraries range from values of ≈ 3 to ≈ 7 indicating that the chain freely explores the state space. The exploratory behaviour is also reflected by the low mean as the chain does not strongly favour high-value solutions, indeed, $\beta = 0.1$ finds a maximum itinerary value of only 7.7 (all maximums reported are calculated on the 10000 iterations). Secondly, the $\beta = 3$ histogram displays a negatively skewed distribution with a higher mean of 6.9. The chain now balances exploration with exploitation (i.e. a transitional regime) visiting itinerary values in a range of ≈ 5 to ≈ 8. This chain now focuses more on high-value itineraries (it has a minimum of ≈ 5 compared to ≈ 3 in the previous histogram) and still explores the state space given that it reaches a better maximum itinerary value of 8.22. Finally, the $\beta = 12$ histogram displays a very narrow distribution ranging from 7.5 to 8.1 achieving the highest average of 8.03. Most of the histogram's 1000 samples are concentrated in a few bins, which clearly shows that the chain becomes "stuck" at high-valued itineraries. Indeed, this exploitation

tendency means the chain loses its ability to explore the state space as it cannot escape good solutions it first encounters, which often are not the global optimum. Indeed, this is notably reflected by the fact that it finds a lower maximum value than $\beta = 3$ at only 8.12. In this case, $\beta = 12$ still gets "lucky" and reaches a better solution on average compared to the others.

4.2

We could improve our solutions by addressing the exploration-exploitation trade-off. Indeed, as per the lecture notes and content for week 9, we can adopt the notion of simulated annealing. Similarly to the process of annealing for materials like metal, we start our β parameter at a low value at the start of our loop and then increase it gradually throughout the $N = 10,000$ iterations. This enables us to start with a β parameter which explores the state space C freely and as β gradually increases we focus on a higher valued subset of the C state space.

There are different ways to conduct this annealing process, the simplest being linear annealing. We decide to approach this process in two different ways:

1. Linearly increasing β at each step by starting at $\beta_{initial}$ and finishing at β_{final} ($\beta_{final} > \beta_{initial}$) by incrementing β by a fixed constant factor at each step: $\frac{(\beta_{final} - \beta_{initial})}{N-1}$ (this ensures β goes to β_{final} after N iterations).
2. Exponentially increasing β at each step starting and ending at the different values and multiplying β by a constant factor: $(\beta_{final}/\beta_{initial})^{1/N}$ at each step. This enables us to explore the state space slowly and then increase sharply towards higher value alternatives.

Given our previous $\beta \in \{0.1, 3, 12\}$ we start our linear annealing at $\beta_{initial} = 0.1$ and increase it to 12 by the $10,000^{th}$ iteration. For our exponential annealing method, we noticed that these inputs did not converge by $N = 10,000$, thus, we started at a higher $\beta_{initial} = 1$ until $\beta_{final} = 30$ which resulted in a similar convergence pattern as the linear annealing (as seen in Figure 5).

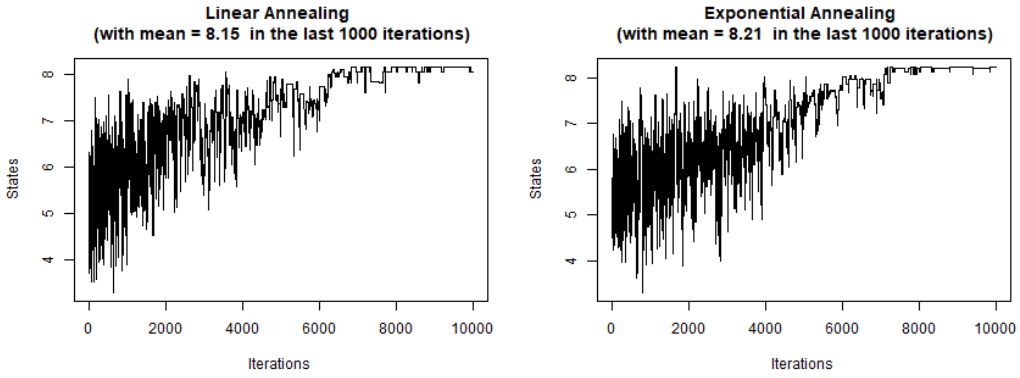

Figure 5: Linear and Exponential Annealing Histograms over 10000 simulations

From the perspective of our algorithm, we implement these two annealing processes by simply modifying a few lines of our previous Q4.1 R script. First, we define the starting and finishing beta for each of our methods (linear and exponential). Afterwards, we pre-compute the constant linear and exponential factors. We then create a new outer loop going through each method and initialise the beta accordingly. This enables us to keep the same inner loop, the only difference being that, at the end of each iteration we increment beta as per the method chosen. Finally, once we exit the inner loop, we create a line plot of the 10000 simulations itinerary values to show that both methods explore the state space and converge in a similar way. We also compute the average of the last 1000 values for comparability with our fixed beta implementation.

As we can see from Figure 5, both annealing methods outperform our fixed β approaches from part (i). The linear annealing method reaches an average of 8.15 with a maximum value of 8.16 while the exponential annealing method has an average of 8.21 with a maximum value 8.23. Although they both exceed the best fixed β average of 8.02 obtained with $\beta = 12$, only the exponential annealing exceeds the maximum value found with $\beta = 3$ of 8.22. This suggests that the exponential annealing method enables for more exploration early on, potentially leading it to find higher maximum itineraries.

Overall, we can explain this improvement in the following way: a high fixed β implies the chain may become "stuck" in a local optimum since it rarely accepts moves to lower-valued states. In contrast, annealing allows the chain to explore the state space freely at the beginning (when β is small) enabling it to discover different regions of the state space. As β increases, the chain gradually focuses on the high-value regions it has discovered. In the end, it converges to slightly better solutions (in this case) on average compared to those found by starting with a fixed high β. Looking at the line plots in Figure 5, we observe that both annealing methods explored the state space. However, compared to linear annealing, the exponential annealing method exhibited more exploration in the initial phase (given that it stays at lower values longer). This may explain why it obtains a better maximum value than the linear annealing method. Finally, although simulated annealing successfully improves the fixed β approach by balancing exploration and exploitation throughout the simulation, we note that the improvement remains marginal which means that our high beta was able to get close to the global optimum early on.