CMA6134

Computational Methods

Trimester 2, 2023/2024

Assignment 1 Report

TT6L_G1

No.	Name	Student ID
1	SAVITHAR A/L RAVICHENDERAN	1211111607
2	TAN JIN YI	1221305792
3	ONG KAI XIN	1231101152
4	NISA NABILAH BINTI MOHD FAIRUZ	1231302349

Car Wash Simulator

1. Introduction

The purpose of this project is to develop a car wash simulator for a car wash center with three wash bays. The simulator models a queuing system where cars line up in a single lane and proceed to an available wash bay. Each car owner may opt for a different car wash service, with service types randomly generated for each owner. This report details the implementation of the simulation, including random number generation for service times and inter-arrival times, and the simulation results.

In the whole code, we have 9 different functions that makes the car wash simulator.

1. main
2. custDetails
3. probGenerator
4. lcg
5. rvge
6. rvgu
7. counter
8. interTable
9. EvalResult

The `main` function simulates a queueing system where `n` customers arrive, queue up for service at one of three counters, and then receive service. The function uses different methods to generate random numbers for inter-arrival and service times, then processes each customer through the system, and display multiple various metrics.

The `custDetails` function generates the necessary details for each customer in a simulation, specifically their inter-arrival times and service times. These details are essential for simulating the queueing system described in the `main` function.

The `probGenerator` function generates a probability distribution based on different random number generation methods and normalizes the generated probabilities to sum up to 1. The purpose of this function is to provide probabilities for events (like inter-arrival times or service times) in the simulation.

The `lcg` function generates a sequence of pseudo-random numbers using the Linear Congruential Generator algorithm. This sequence can be used in the simulations that requires random numbers.

The `rvge` function generates random numbers following an exponential distribution, which can be useful for modeling time between events in a Poisson process (e.g., inter-arrival times in a queueing system).

The `rvgu` function generates a sequence of random numbers following a uniform distribution within the interval [0, 1]. This can be useful for various applications where uniformly distributed random numbers are needed.

The `counter` function is designed to compute and display the cumulative distribution function (CDF) and corresponding ranges for service times based on provided probabilities. The function also updates a matrix `rangeCounter` with the calculated ranges for a specific counter. This can be useful in simulations where service times need to be allocated based on probabilities.

The `interTable` function calculates probabilities, CDF, and ranges for inter-arrival times using different random number generators, normalizes these values, and updates the `rangeArrival` array. It also displays the inter-arrival probability table.

The `evalResults` function calculates and prints several key performance metrics for a queuing system simulation. These metrics are derived from the inter-arrival times and service times of customers. The function computes average values, waiting times, and probabilities, which help in understanding the overall performance of the system.

2. Simulation Details

2.1 Random Number Generators:

- We used MATLAB's built-in `rand` function to generate random numbers. The user can choose the type of random number generator before the simulation begins. Theres 4 type provided:
    1. Simple Random Function
    2. Linear Congruential Generators Formula
    3. Random Variate Generator for Exponential Distribution
    4. Random Variate Generator for Uniform Distribution

Formulas Used in the System:

```
if r == 1
    for i=1:loopNum
        probability(i) = rand();
        % normalising so that sum (probability) = 1
        probability = probability / sum(probability);
        probability = round(probability * 100) / 100;
    end
```

Code (in probGenerator function): Random number generator choice 1 is Simple Random Function

The formula for the 1st choice is simple random number function. When user input 1, the system would generate random numbers. The `rand` function is to generate random numbers between 0 and 1 for each element in the `probability` array. It then normalizes the probabilities so that their sum is 1 and rounds them to two decimal places.

```
function output=lcg (loopNum,m,a, c)
    seed = rand();
    x = seed;

    n = zeros(1, loopNum);

    for i=1:loopNum
        n(i) = mod(a*x+c,m);
        x=n(i);
    end
    output =n;
```

Code: Random number generator choice 2 is Linear Congruential Generators (LCG) Formula

The formula for the 2nd choice of random number generator is Linear Congruential Generators. It is defined by the parameters 'm','a', and 'c', and the number of random numbers to generate is specified by 'loopNum'.

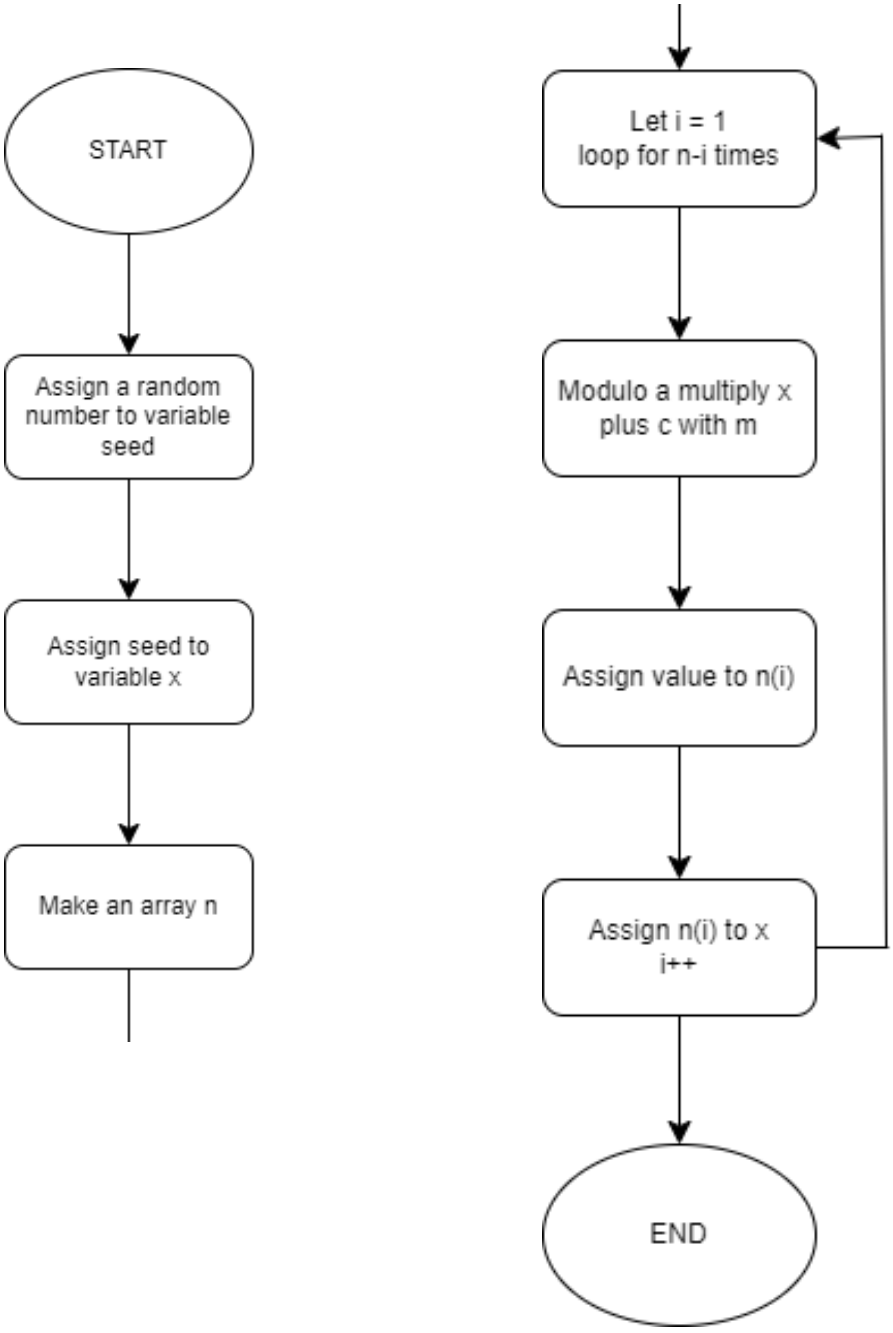
For starters, `seed` is initialized using `rand` function, which generates a random number between 0 and 1. This seed is the initial value for the LCG.

- `x` is set to the seed value.
- `n` is initialized as an array of zeros with a length of `loopNum` to store the generated numbers.

The for loop then runs `loopNum` times to generate the sequence of random numbers. In each iteration, the next number in the sequence is calculated using the LCG formula: `n(i) = mod(a * x + c, m)`, where `a` is the multiplier, `c` is the increment, `m` is the modulus, and `x` is the current number in the sequence. After calculating `n(i)`, `x` is updated to the newly calculated value to be used in the next iteration.

The generated sequence of numbers `n` is assigned to the output variable.

Flowchart:


START
Assign a random number to variable seed
Assign seed to variable x
Make an array n
Let i = 1
loop for n-i times
Modulo a multiply x plus c with m
Assign value to n(i)
Assign n(i) to x
i++
END

```
function output = rvge(loopNum)
    sequence = rand(1,loopNum) ;
    x = zeros(1,loopNum) ;

    for i=1:length(sequence)
        x(i) = (-1/1)*(log(1-sequence(i)));
    end
    output = x;
end
```

Code: Random number generator choice 3 is Random Variate Generator for Exponential Distribution

The formula for the 3rd choice is random variate generator for exponential distribution. The rvge function generates random numbers following an exponential distribution, which can be used for modeling time between events in a Poisson process such as inter-arrival times in a queueing system.

The `sequence` is initialized with `loopNum` random numbers generated with `rand` function, which produces values uniformly distributed between 0 and 1. `x` is initialized as an array of zeros with a length of `loopNum` to store the generated exponential random variates.

The for loop iterates over each element in the `sequence` array. In each iteration, it applies the inverse transform sampling method to generate an exponential random variate:

- The expression `(-1/1) * log(1 - sequence(i))` computes the exponential random variate.
- `(-1/1)` is the reciprocal of the rate parameter ($\lambda = 1$ for this function).
- `log(1 - sequence(i))` transforms the uniformly distributed random number `sequence(i)` into an exponentially distributed random variate using the inverse of the exponential cumulative distribution function (CDF).

The generated sequence of exponential random variates `x` is assigned to the output variable.

Mathematical Formula

The formula `(-1/1) * log(1 - sequence(i))` is used to generate values from an exponential distribution with a mean (or rate parameter) of 1.

Exponential distribution derivation

For an exponential distribution with rate parameter λ (where $\lambda=1$ in this case), the probability density function is given by: $f(x)=\lambda e^{-\lambda x}$. The cumulative distribution function (CDF) of an exponential distribution is $F(x)=1-e^{-\lambda x}$. Given a uniform random variable U in the interval (0, 1), the inverse transform sampling method can be used to generate exponential random variables. By setting the CDF equal to U: $U=1-e^{-\lambda x}$

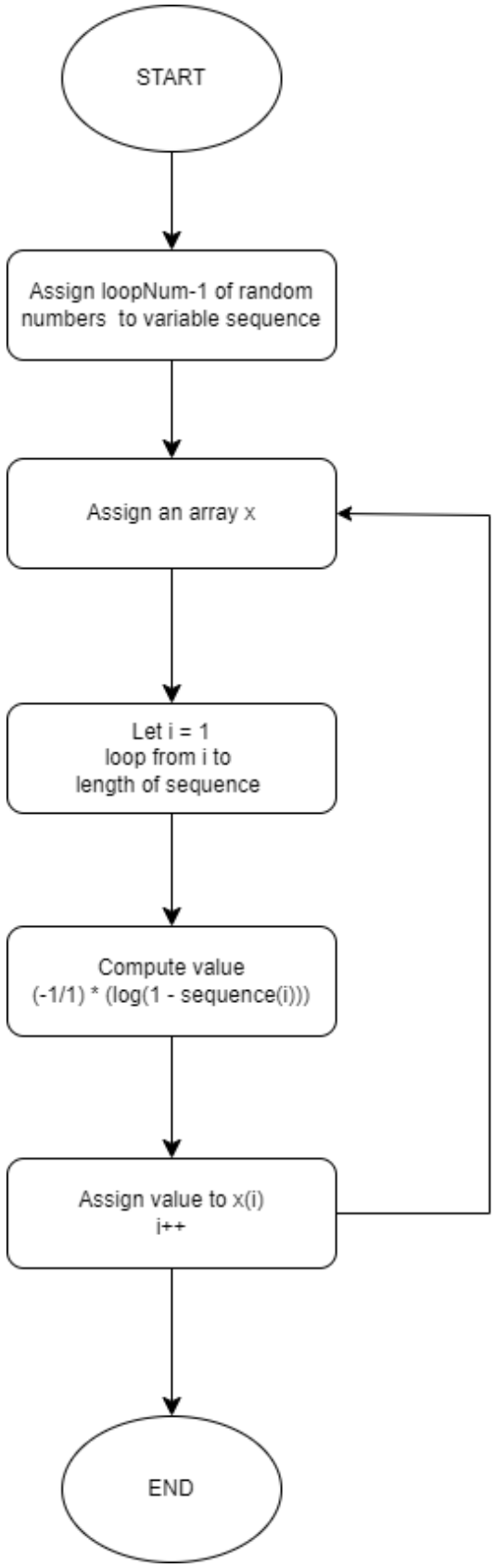
Solving for x: $e^{-\lambda x} = 1-U$

$-\lambda x=\log(1-U)$ Since $\lambda=1$

$x=(-1/\lambda)\ (\log(1-U))$ $x=-\log(1-U)$

This is exactly the transformation applied in the code to convert uniform random variables into exponential random variables with rate parameter 1.

Flowchart:


START
Assign loopNum-1 of random numbers to variable sequence
Assign an array x
Let i = 1
loop from i to
length of sequence
Compute value
(-1/1) * (log(1 - sequence(i)))
Assign value to x(i)
i++
END

```
function output = rvgu(loopNum)
    %interval a is 0
    %interval b is 1

    sequence = rand(1, loopNum) ;
    x = zeros(1,loopNum);

    for i=1:length(sequence)
        x(i) = 0 + (1 - 0)*sequence(i);
    end
    output = x;
```

Code: Random number generator choice 4 is Random Variate Generator for Uniform Distribution

The formula for the 4th choice is random variate generator for uniform distribution. The `rvgu` function generates a sequence of random numbers following a uniform distribution within the interval [0, 1]. This is useful where uniformly distributed random numbers are needed.

The `sequence` is initialized with `loopNum` random numbers generated using `rand` function, which produces values uniformly distributed between 0 and 1.

The `x` is initialized as an array of zeros with a length of `loopNum` to store the generated uniform random variates. The for loop iterates over each element in the `sequence` array. In each iteration, it assigns the value of `sequence(i)` to `x(i)`. The formula `0 + (1 - 0) * sequence(i)` simplifies to just `sequence(i)` since the interval [a, b] is [0, 1].

The generated sequence of uniform random variates `x` is assigned to the output variable.

Mathematical Formula

The core of this function lies in the formula:

$x(i) = a + (b - a) \cdot sequence(i)$

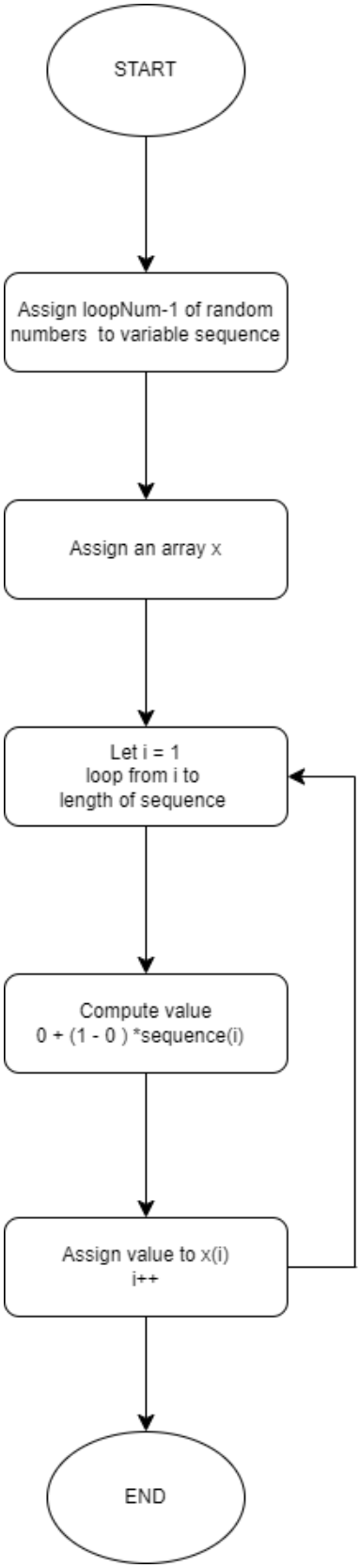
Where a=0 and b= 1, so the formula simplifies to:

$x(i) = sequence(i)$

This formula generates random numbers uniformly distributed in the interval [a,b]. Since a=0 and b=1, the generated random numbers are uniformly distributed in the interval [0,1].

The function `rvgu` generates `loopNum` random numbers uniformly distributed in the interval `[0,1]`. It does this by generating `loopNum` uniform random numbers using MATLAB's `rand` function, which produces values in the interval `[0,1]`. The formula used, $a + (b - a) \cdot sequence(I,$ is a general formula for scaling and translating uniform random numbers to any desired interval `[a,b]`. In this case, since a=0 and b=1, the formula simplifies to just the uniform random numbers themselves.

Flowchart:


START
Assign loopNum-1 of random numbers to variable sequence
Assign an array x
Let i = 1
loop from i to length of sequence
Compute value
0 + (1 - 0) *sequence(i)
Assign value to x(i)
i++
END

2.2 probGenerator function:

The function `probGenerator` generates a probability distribution based on different random number generation methods specified by the input `r`. The distribution is normalized so that the sum of the probabilities equals 1.

```
function [probability] = probGenerator(r, loopNum, m, a, c)

    probability = zeros(1, loopNum);

    if r == 1
        for i=1:loopNum
            probability(i) = rand();
            % normalising so that sum (probability) = 1
            probability = probability / sum(probability);
            probability = round(probability * 100) / 100;
        end

    elseif r == 2
        probability= lcg(loopNum,m, a, c);

    elseif r == 3
        probability = rvge(loopNum);

    elseif r == 4
        probability = rvgu(loopNum);

    else
        fprintf('zero \n');
        return
        end
```

The code first initializes the probability vector with zeros. Then, depending on the value of r, the function generates probabilities using different methods:

- ‘1’ for built-in random number generator (rand).
- ‘2’ for a linear congruential generator (LCG).
- ‘3’ for a random variate generator (Exponential Distribution).
- ‘4’ for another random variate generator (Uniform Distribution).
- If ‘r’ does not match any of these, it prints 'zero' and exits.

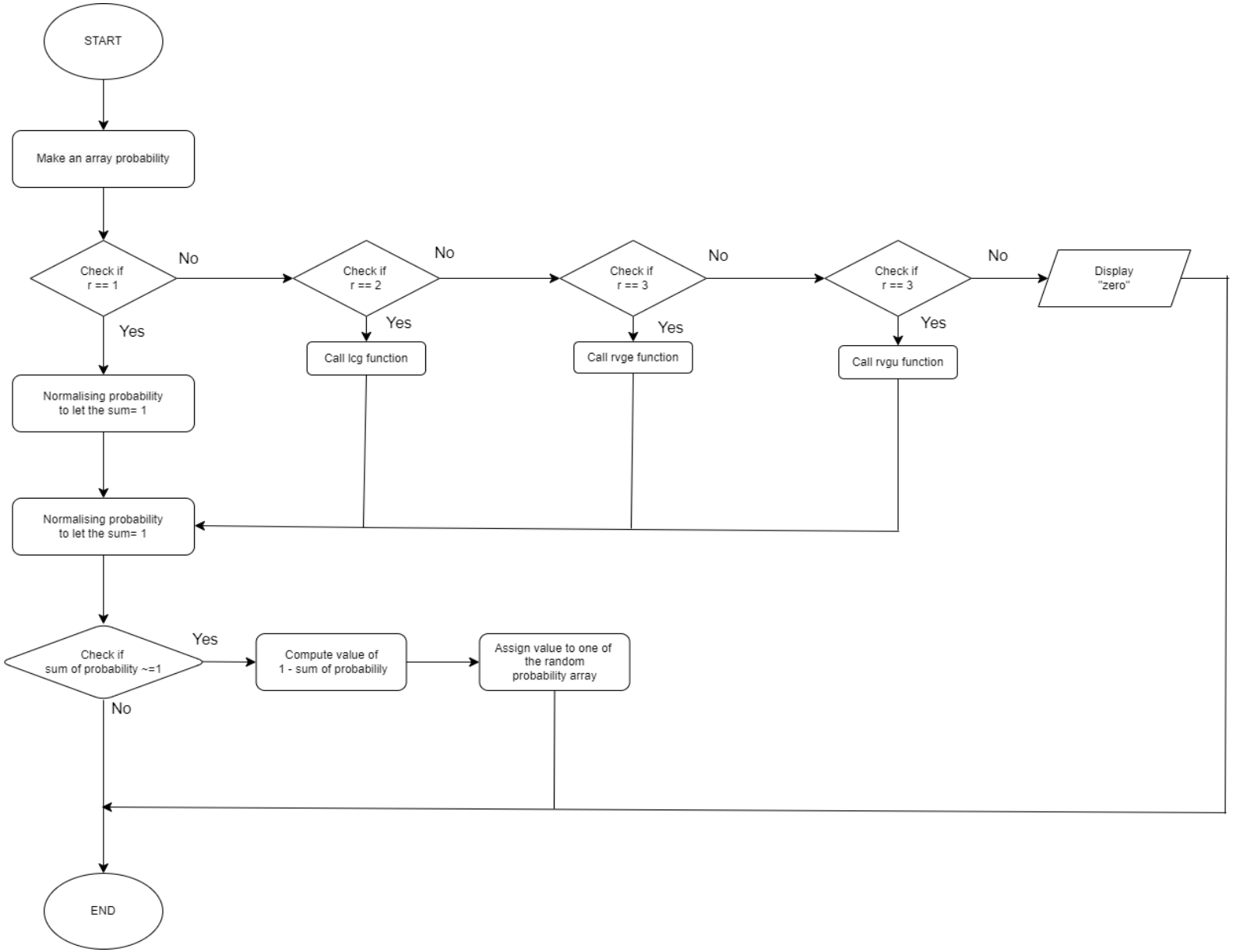
```
% normalising so that sum sum(probability) = 1
probability = probability / sum(probability);
probability = round(probability * 100) / 100;

% sometimes sum (probability) will not equal 1 because each value is rounded
% so here the leftover is added to a random value in probability
if sum(probability) ~= 1
    leftover = 1 -sum(probability);
    index = randi(1, loopNum) ;
    probability(index) = probability(index) + leftover;
end

% returns probability
output = probability;
```

It then normalizes the generated probabilities, so their sum equals 1and rounds each probability to two decimal places. Due to rounding, the sum of the probabilities might not be exactly 1. This segment adjusts for such discrepancies:

- Calculates the leftover amount that needs to be added.
- Randomly selects an index in the probability vector.
- Adds the leftover amount to the randomly chosen index to ensure the total sum is exactly 1.

Lastly, it sets the final normalized and adjusted probability vector as the output of the function.

Flowchart:


START
Make an array probability
Check if r == 1
No
Yes
Check if r == 2
No
Yes
Call lcg function
Check if r == 3
No
Yes
Call rvge function
Check if r == 3
No
Yes
Call rvgu function
Display "zero"
Normalising probability to let the sum= 1
Normalising probability to let the sum= 1
Check if sum of probability ~=1
Yes
No
Compute value of 1 - sum of probabilily
Assign value to one of the random probability array
END

2.3 custDetails function:

The `custDetails` function generates the necessary details for each customer in a simulation, specifically their inter-arrival times and service times. These details are essential for simulating the queueing system described in the `main` function.

```
function [intArrival, service] = custDetails(n, r, loopNum, m, a, c)
   % This function generates random details for n customers

    intArrival = zeros(1, n);
    intArrival(1) = 0;
    service = zeros(1, n);
```

Initializes the arrays `intArrival` and `service` with zeros. The first inter-arrival time is set to 0, indicating that the first customer arrives at time 0.

```
if r == 1
    % Generate random number for inter-arrival time n-1 times and assign to the array 'intArrival'
    for i = 2:n
        intArrival(i) = randi([1, 100]);
    end

    % Generate random number for service time n times and assign to the array 'service'
    for i = 1:n
        service(i) = randi([1, 100]);
    end
```

Uses MATLAB's `randi` function to generate random integers between 1 and 100 for inter-arrival times (for the 2nd to nth customer) and service times (for all customers).

```
elseif r == 2
    intArrivalLcg = lcg(loopNum, m, a, c);
    for i = 2:n
        intArrival(i) = round(intArrivalLcg(i) * 99) + 1;
    end

    serviceTimeLcg = lcg(loopNum, m, a, c);
    for i = 1:n
        service(i) = round(serviceTimeLcg(i) * 99) + 1;
    end
```

Uses a Linear Congruential Generator (LCG) to generate random numbers. The LCG function `lcg` generates a sequence of numbers, which are then scaled and rounded to integers between 1 and 100 for inter-arrival and service times.

```
elseif r == 3
    for i = 2:n
        intArrival(i) = mod(round(rvge(loopNum) * 99), 100) + 1;
    end
    for i = 1:n
        service(i) = mod(round(rvge(loopNum) * 99), 100) + 1;
    end
```

Uses a custom exponential distribution generator function `rvge` to generate random numbers. The generated numbers are scaled, rounded, and mapped to integers between 1 and 100.

```
elseif r == 4
    for i = 2:n
        intArrival(i) = mod(round(rvgu(loopNum) * 99), 100) + 1;
    end
    for i = 1:n
        service(i) = mod(round(rvgu(loopNum) * 99), 100) + 1;
    end
```

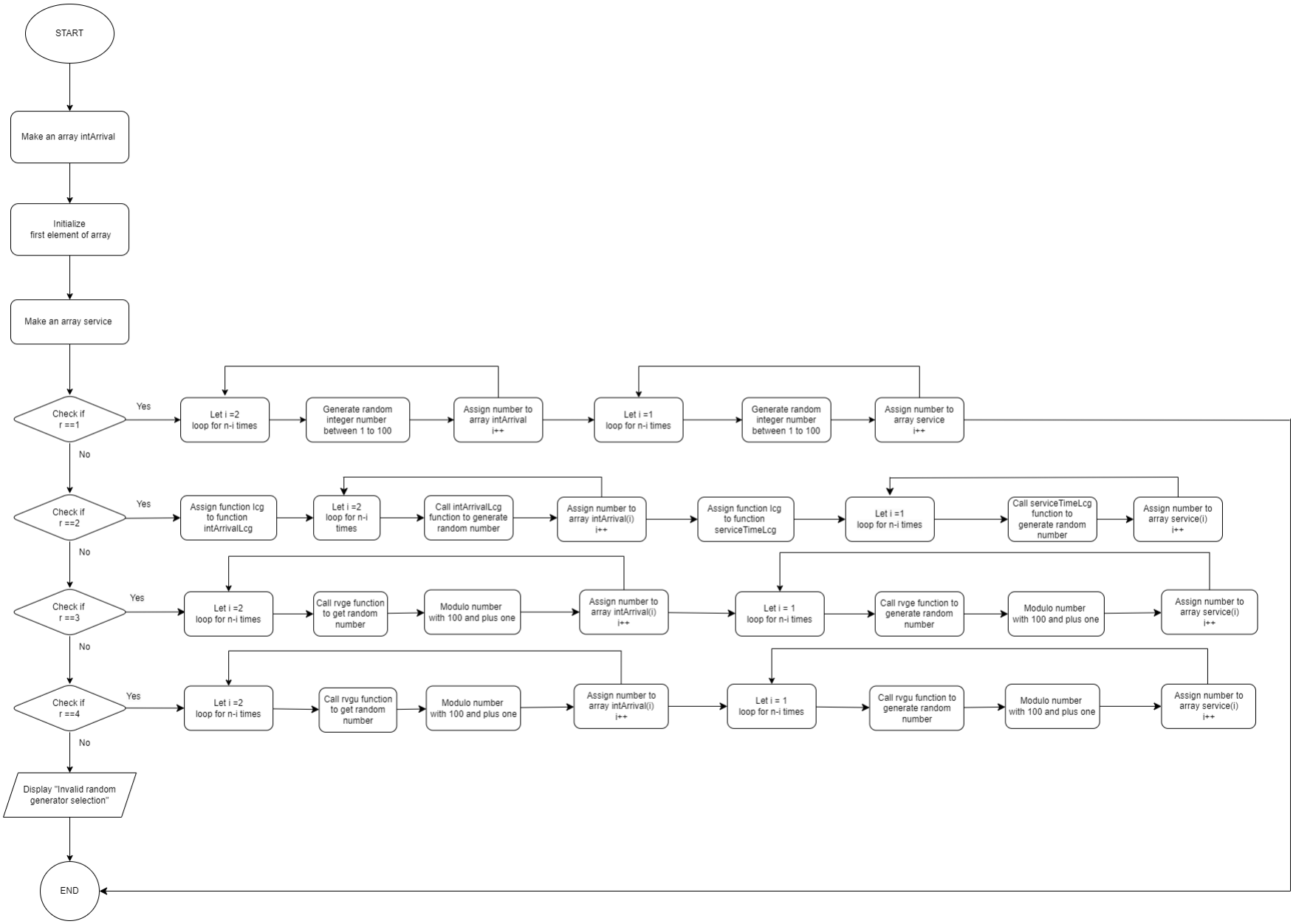
Uses a custom uniform distribution generator function `rvgu` to generate random numbers. Like the previous method, the generated numbers are scaled, rounded, and mapped to integers between 1 and 100.

```
    else
        fprintf('Invalid random generator selection\n');
        return
    end
end
```

Handles invalid random generator selection by printing an error message and terminating the function.

The `custDetails` function generates inter-arrival and service times for `n` customers using one of four specified methods for random number generation. This data is then used in the `main` function to simulate the queueing system, determining when each customer arrives and how long they will be serviced.

Flowchart:


START
Make an array intArrival
Initialize
first element of array
Make an array service
Check if
r ==1
Yes
Let i =2
loop for n-i times
Generate random
integer number
between 1 to 100
Assign number to
array intArrival
i++
Let i =1
loop for n-i times
Generate random
integer number
between 1 to 100
Assign number to
array service
i++
No
Check if
r ==2
Yes
Assign function lcg
to function
intArrivalLcg
Let i =2
loop for n-i
times
Call intArrivalLcg
function to generate
random number
Assign number to
array intArrival(i)
i++
Assign function lcg
to function
serviceTimeLcg
Let i =1
loop for n-i times
Call serviceTimeLcg
function to
generate random
number
Assign number to
array service(i)
i++
No
Check if
r ==3
Yes
Let i =2
loop for n-i times
Call rvge function
to get random
number
Modulo number
with 100 and plus one
Assign number to
array intArrival(i)
i++
Let i = 1
loop for n-i times
Call rvge function to
generate random
number
Modulo number
with 100 and plus one
Assign number to
array service(i)
i++
No
Check if
r ==4
Yes
Let i =2
loop for n-i times
Call rvgu function
to get random
number
Modulo number
with 100 and plus one
Assign number to
array intArrival(i)
i++
Let i = 1
loop for n-i times
Call rvgu function to
generate random
number
Modulo number
with 100 and plus one
Assign number to
array service(i)
i++
No
Display "Invalid random
generator selection"
END

2.4 main function:

The `main` function simulates a queueing system where `n` customers arrive, queue up for service at one of three counters, and then receive service. The function uses different methods to generate random numbers for inter-arrival and service times, then processes each customer through the system, recording and displaying various metrics.

1. **Initialization**: The function accepts parameters to control the number of customers (`n`), the random number generation method (`r`), the number of iterations (`loopNum`), and parameters for the linear congruential generator (`m`, `a`, `c`).
2. **Customer Details Generation**: It calls `custDetails` to generate inter-arrival and service times for each customer based on the selected random number generator.
3. **Service Time Probability Tables**: It generates service time probability tables for each of the three counters using `probGenerator` and `counter`.
4. **Inter-Arrival Time Table**: It creates a probability table for inter-arrival times using `interTable`.
5. **Customer Arrival Simulation**: It simulates customer arrivals and assigns them to queues at different counters based on the number of items they have.
6. **Queue Assignment**: Customers are assigned to the shortest queue or a dedicated queue if they have fewer items.
7. **Service Time Determination**: The function determines the service time for each customer by comparing random values against the pre-calculated service time ranges using `findServiceTime`.
8. **Service Time Adjustment**: It adjusts the service start and end times for each customer to ensure no overlap at each counter.
9. **Statistics Calculation**: It calculates and prints various performance metrics such as waiting times, time spent in the system, and probabilities of waiting.
10. **Results Display**: The function displays detailed results for each counter and overall system performance by calling `printCounterResults` and `evalResults`.

```
function output = main(n, r, loopNum, m, a, c)
    fprintf('\n********** WELCOME TO CMA6134 CARWASH SIMULATOR **********\n');
    fprintf('\nRandom Generators: 1: rand(), 2: LCG, 3: RVGE, 4: RVGU\n');
    fprintf('\n');
    n = input('Enter the # of customers : ');
    r = input('Choose a random generator: ');
    loopNum = 6;
    m = [];
    a = [];
    c = [];
    if r == 2 % if LCG is chosen, prompt for LCG parameters
        m = input('Enter the modulus (m): ');
        a = input('Enter the multiplier (a): ');
        c = input('Enter the additive constant (c): ');
    end

 [interArrival, svcTime] = custDetails(n, r, loopNum, m, a, c); % passing two arrays from custDetails
```

Displays a welcome message. Prompts the user to enter the number of customers (`n`) and select a random generator (`r`). If LCG is chosen, additional parameters (`m`, `a`, `c`) are requested. Calls `custDetails` to **get arrays for inter-arrival times and service times** for `n` customers using the specified random number generation method.

```
 % array of ranges generated for service time
 rangeCounter = zeros(3, loopNum);

 fprintf('\n---------- Pre-Defined Tables ----------\n');
 for counterNum = 1:3
     probability = probGenerator(r, loopNum, m, a, c);
     rangeCounter(counterNum, :) = counter(loopNum, counterNum, probability, rangeCounter);
 end

 % array of ranges generated for inter-arrival time
 rangeArrival = zeros(1, 4);
 rangeArrival = interTable(r, loopNum, m, a, c, rangeArrival);
```

Generates and stores ranges for service times for each of the three counters using `probGenerator`. Then, **generates ranges for inter-arrival times**.

```
 numOfItems = randi([1, 10], 1, n);

 custArrival = zeros(1, n); % inter-arrival time for each car

 clock = 0; % set timer
 custArrival(1) = clock; % car 1 arrives at time 0

 % comparing RN to inter-arrival ranges
 for i = 2:n
     if interArrival(i) >= 0 && interArrival(i) <= rangeArrival(1)
         custArrival(i) = 1;
     elseif interArrival(i) >= rangeArrival(1) + 1 && interArrival(i) <= rangeArrival(2)
         custArrival(i) = 2;
     elseif interArrival(i) >= rangeArrival(2) + 1 && interArrival(i) <= rangeArrival(3)
         custArrival(i) = 3;
     elseif interArrival(i) >= rangeArrival(3) + 1 && interArrival(i) <= rangeArrival(4)
         custArrival(i) = 4;
     else
         fprintf('error\n');
     end
 end
```

Initializes arrays to store the number of items each customer has and their arrival times. Sets the clock to 0 for the first customer's arrival time. **Determines the inter-arrival time** for each customer based on the generated ranges.

```
clockRecord = zeros(1, n);
clockRecord(1) = 0; % record first arrival time as 0

queue1 = []; % arrays for queues of each counter
queue2 = [];
queue3 = [];

fprintf('\n---------- Simulation Logs ----------\n\n');
for i = 1:n
    if numOfItems(i) <= 3
        queue3(end+1) = i; % add car to queue 3
        disp(['Car ', num2str(i), ' arrives at ', num2str(clockRecord(i)), ' and queues at Counter 3']);
    elseif numel(queue1) <= numel(queue2)
        queue1(end+1) = i; % add car to queue 1
        disp(['Car ', num2str(i), ' arrives at ', num2str(clockRecord(i)), ' and queues at Counter 1']);
    else
        queue2(end+1) = i; % add car to queue 2
        disp(['Car ', num2str(i), ' arrives at ', num2str(clockRecord(i)), ' and queues at Counter 2']);
    end
end
```

Records and prints the arrival times for each customer, updating the clock for each subsequent customer's arrival time. **Assigns each customer to a queue** based on the number of items they have. Customers with 3 or fewer items go to queue 3, and the rest go to either queue 1 or queue 2, whichever is shorter. Essentially, initializes queue arrays and logs the arrival of each car, assigning them to the appropriate queue based on the number of items they have.

```
custServ = zeros(1, n);
location = zeros(1, n); % use to check location of customer

% finding location of each customer
for i = 1:n
    if any(queue1 == i)
        location(i) = 1;
    elseif any(queue2 == i)
        location(i) = 2;
    elseif any(queue3 == i)
        location(i) = 3;
    else
        fprintf('error\n');
    end
end

timeSvcBegins = zeros(1, n);
timeSvcEnds = zeros(1, n);

% finding RN in the ranges generated for each counter's service time
for i = 1:n
    if location(i) == 1
        custServ(i) = findServiceTime(svcTime(i), rangeCounter(1, :));
    elseif location(i) == 2
        custServ(i) = findServiceTime(svcTime(i), rangeCounter(2, :));
    elseif location(i) == 3
        custServ(i) = findServiceTime(svcTime(i), rangeCounter(3, :));
    else
        fprintf('error hahaha\n');
    end

    if custServ(i) > 0
        timeSvcBegins(i) = clockRecord(i);
        timeSvcEnds(i) = timeSvcBegins(i) + custServ(i);
    end
end
```

Determines the service time for each customer based on their location and the pre-generated ranges. Updates the times when service begins and ends.

```
% Adjusting service times for queue 1
for i = 2:numel(queue1)
    if timeSvcBegins(queue1(i)) < timeSvcEnds(queue1(i-1))
        timeSvcBegins(queue1(i)) = timeSvcEnds(queue1(i-1));
        timeSvcEnds(queue1(i)) = timeSvcBegins(queue1(i)) + custServ(queue1(i));
    end
end

% Adjusting service times for queue 2
for i = 2:numel(queue2)
    if timeSvcBegins(queue2(i)) < timeSvcEnds(queue2(i-1))
        timeSvcBegins(queue2(i)) = timeSvcEnds(queue2(i-1));
        timeSvcEnds(queue2(i)) = timeSvcBegins(queue2(i)) + custServ(queue2(i));
    end
end

% Adjusting service times for queue 3
for i = 2:numel(queue3)
    if timeSvcBegins(queue3(i)) < timeSvcEnds(queue3(i-1))
        timeSvcBegins(queue3(i)) = timeSvcEnds(queue3(i-1));
        timeSvcEnds(queue3(i)) = timeSvcBegins(queue3(i)) + custServ(queue3(i));
    end
end
```

Adjusts the service times for each queue to ensure that customers are serviced sequentially without overlapping times

```
% calculating waiting time and time spent
waitingTime = timeSvcBegins - clockRecord;
timeSpend = waitingTime + custServ;

% displaying departure messages
fprintf('\n');
for i = 1:length(timeSvcEnds)
    disp(['Departure of car ', num2str(i), ' at ', num2str(timeSvcEnds(i))]);
end

% displaying service begin messages
fprintf('\n');
for i = 1:length(timeSvcBegins)
    disp(['Service of car ', num2str(i), ' begins at ', num2str(timeSvcBegins(i))]);
end
```

Calculates the waiting time and total time spent for each customer. **Prints departure and service** begin times.

```
% Displaying detailed results for each counter
fprintf('\n---------- Simulation Tables ----------\n\n');

svcTypes = {'Express', 'Economic', 'Full Service'};


fprintf('Overall Simulation Table:\n');
fprintf('| n  | RN | Inter-arrival time | Arrival time | Number of items | Service Type    |\n');
fprintf('| %-2d | -- | %-18d | %-12d | %-15d | %-15s |\n', 1, 0, 0, numOfItems(1), svcTypes{randi(3)});

clock = clock + custArrival(2); % clock is set to car 2's inter-arrival time
clockRecord(2) = clock; % record second arrival time
```

The initial `fprintf` statement prints a header indicating the start of the simulation tables section and header row of overall simulation table. `svcTypes` is an array of strings representing the types of services available: 'Express', 'Economic', and 'Full Service'. For the first customer, the table prints their details. Since it's the first customer, the inter-arrival time and arrival time are both zero. A random service type is chosen from `svcTypes`. The `clock` is updated to the inter-arrival time of the second customer and recorded in `clockRecord`.

```
for i = 2:n-1
    svcType = svcTypes{randi(3)};
    fprintf('| %-2d | %-2d | %-18d | %-12d | %-15d | %-15s |\n', i, interArrival(i), custArrival(i), clock, numOfItems(i), svcType);
    clockRecord(i) = clock;
    clock = clock + custArrival(i+1);
end
```

Loop through the remaining customers The loop iterates over the remaining customers (from the second to the second-to-last customer).

```
svcType = svcTypes{randi(3)};
fprintf('| %-2d | %-2d | %-18d | %-12d | %-15d | %-15s |\n', n, interArrival(n), custArrival(n), clock, numOfItems(n), svcType);
clockRecord(n) = clock; % record clock for last car
```

A random service type is chosen for the last customer from the `svcTypes` array. The details of the last customer are printed, including customer number (`n`), inter-arrival time (`interArrival(n)`), arrival time (`custArrival(n)`), current clock value (`clock`), number of items (`numOfItems(n)`), and service type (`svcType`) and the arrival time of the last customer is recorded in `clockRecord`.

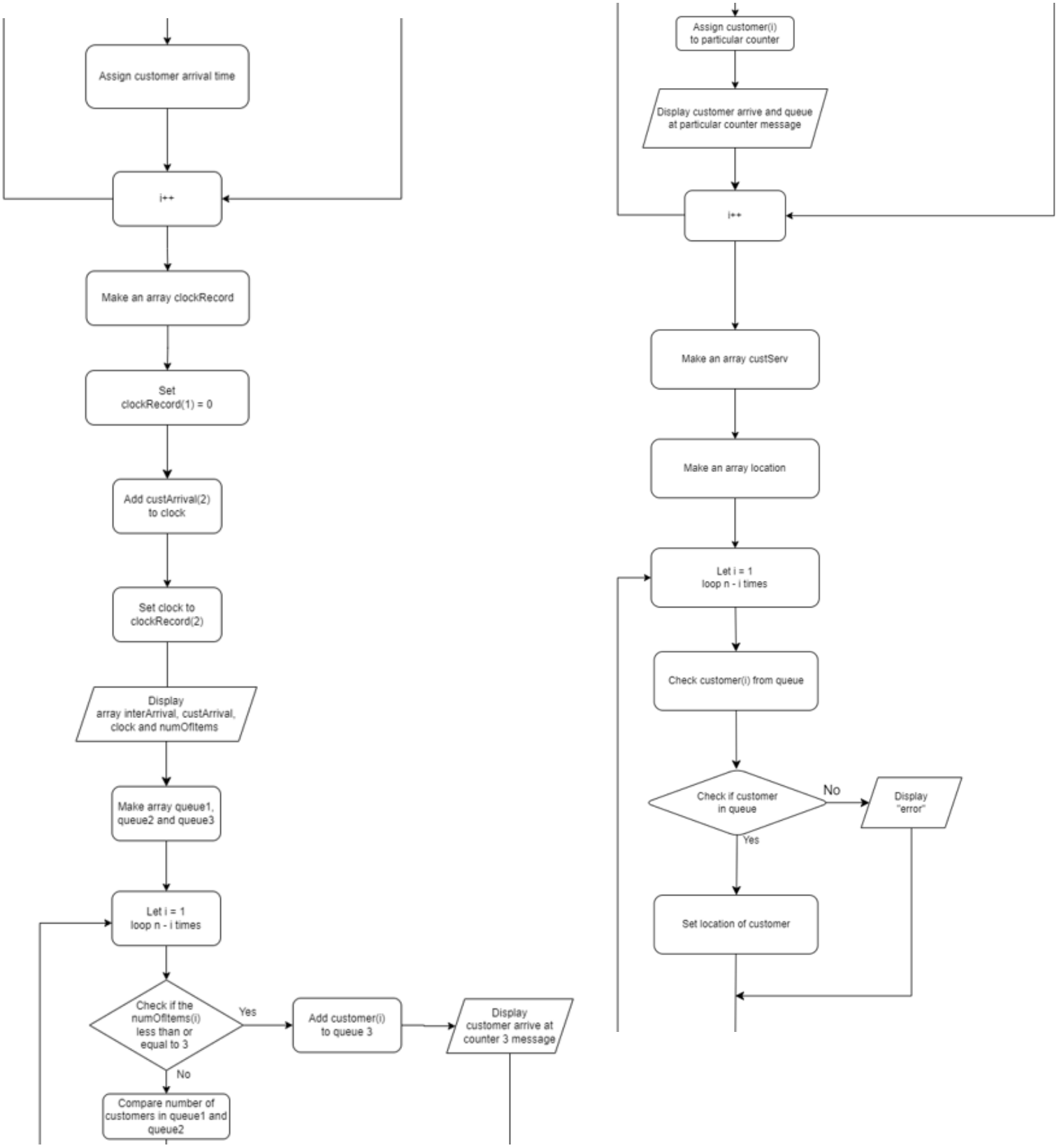
```
    fprintf('\n');

    printCounterResults('Wash Bay 1:', queue1, svcTime, custServ, timeSvcBegins, timeSvcEnds, waitingTime, timeSpend);
    printCounterResults('Wash Bay 2:', queue2, svcTime, custServ, timeSvcBegins, timeSvcEnds, waitingTime, timeSpend);
    printCounterResults('Wash Bay 3:', queue3, svcTime, custServ, timeSvcBegins, timeSvcEnds, waitingTime, timeSpend);

    fprintf('\n');
    evalResults(interArrival, svcTime);

    fprintf('\n********** CARWASH SIMULATION COMPLETED. **********');
    fprintf('\n********** WE HOPE YOU ENJOY OUR SERVICE. **********\n');
end
```

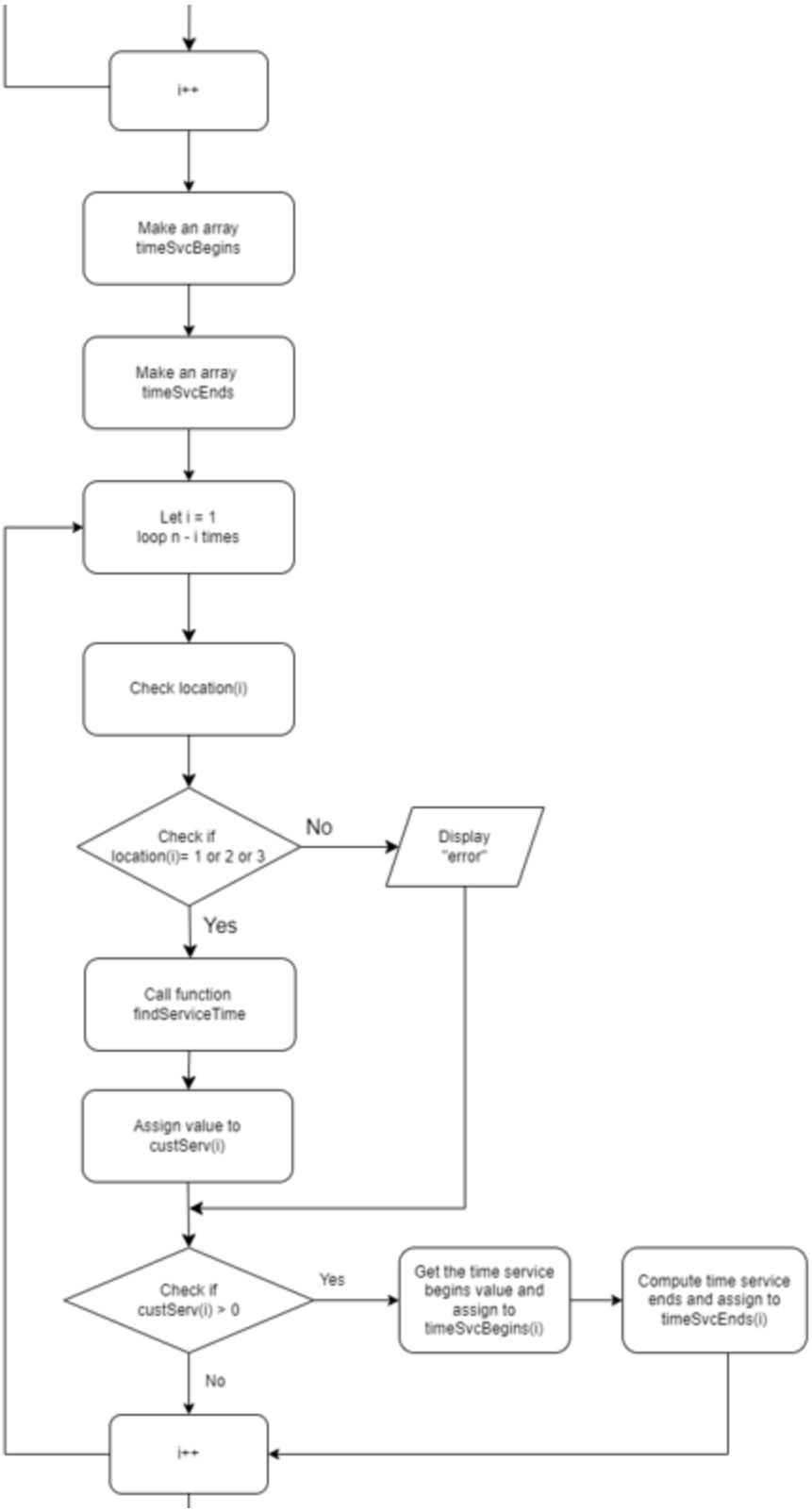
The `printCounterResults` function is called three times to print detailed results for each of the three counters (Wash Bay 1, Wash Bay 2, and Wash Bay 3). Each call to `printCounterResults` includes the counter name and the respective arrays (`queue1`, `queue2`, `queue3`, `svcTime`, `custServ`, `timeSvcBegins`, `timeSvcEnds`, `waitingTime`, `timeSpend`) that hold the service information for each customer.

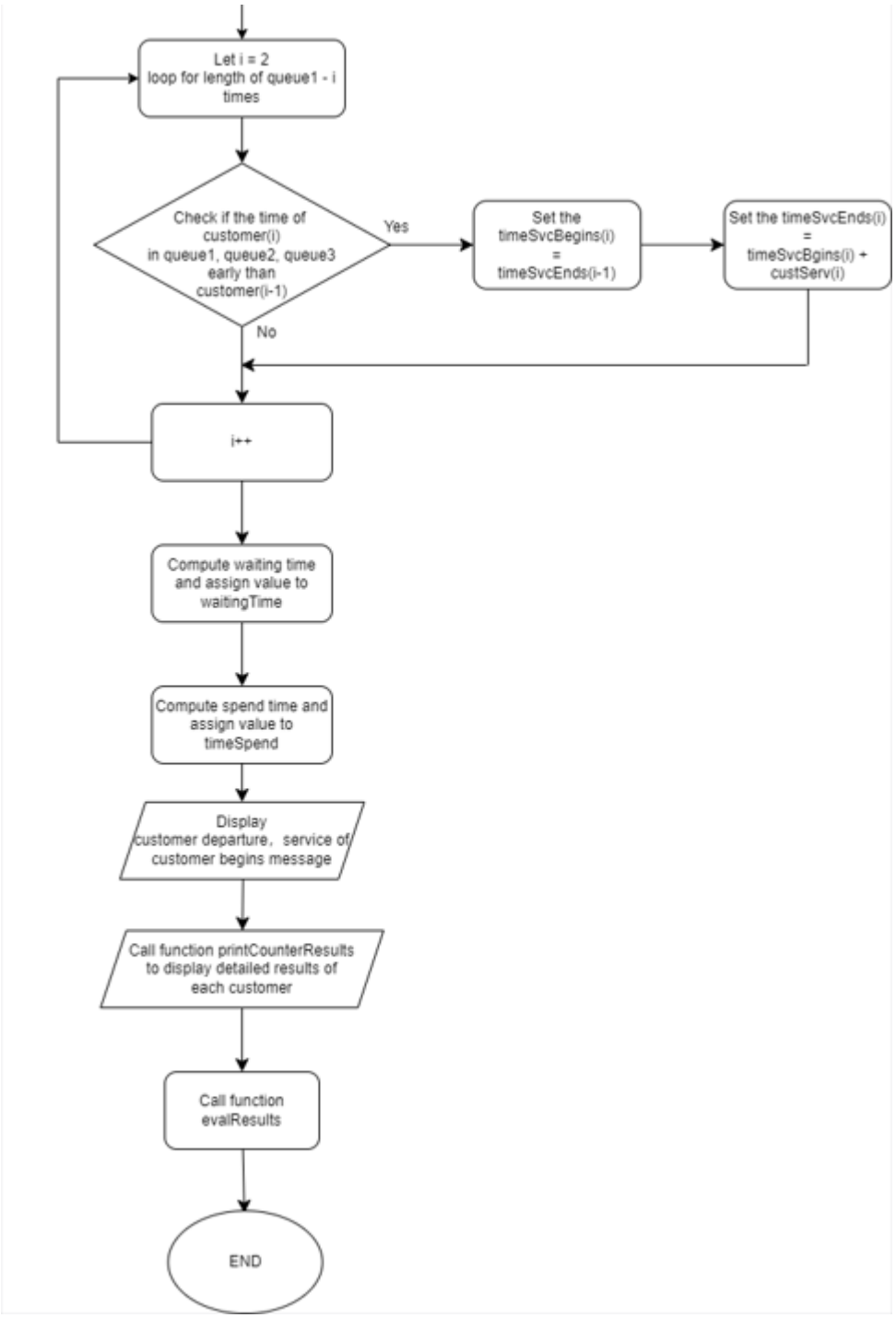
Last but not least, the `evalResults` function is called to evaluate the overall results of the simulation. This function performs calculations or assessments based on the inter-arrival times and service times to provide an overall analysis of the simulation's performance. The final `fprintf` statements print messages indicating that the carwash simulation is completed and expressing hope that the service was enjoyed.

Flowchart:


START
Input parameters
n, r, loopNum, m, a ,c
Call function custDetails
Make an array rangeCounter
Let counterNum = 1
loop 3 - counterNum times
Calling function probGenerator and assign value to probability
Calling function counter and assign value to rangeCounter(counterNum, :)
i++
Make an array rangeArrival
Calling interTable function and assign value to rangeArrival
Make an array numOfItems
Make an array customer
Make an array custArrival
Set clock = 0
Assign clock value to custArrival(1)
Let i = 2
loop n - i times
Check the range of interval arrival time
Check if the interval arrival time out of range
Yes
Display "error"
No


Assign customer arrival time
i++
Make an array clockRecord
Set
clockRecord(1) = 0
Add custArrival(2)
to clock
Set clock to
clockRecord(2)
Display
array interArrival, custArrival,
clock and numOfItems
Make array queue1,
queue2 and queue3
Let i = 1
loop n - i times
Check if the
numOfItems(i)
less than or
equal to 3
Yes
Add customer(i)
to queue 3
Display
customer arrive at
counter 3 message
No
Compare number of
customers in queue1 and
queue2
Assign customer(i)
to particular counter
Display customer arrive and queue
at particular counter message
i++
Make an array custServ
Make an array location
Let i = 1
loop n - i times
Check customer(i) from queue
Check if customer
in queue
No
Display
"error"
Yes
Set location of customer


i++
Make an array timeSvcBegins
Make an array timeSvcEnds
Let i = 1 loop n - i times
Check location(i)
Check if location(i)= 1 or 2 or 3
No
Display "error"
Yes
Call function findServiceTime
Assign value to custServ(i)
Check if custServ(i) > 0
Yes
Get the time service begins value and assign to timeSvcBegins(i)
Compute time service ends and assign to timeSvcEnds(i)
No
i++


Let i = 2
loop for length of queue1 - i times
Check if the time of customer(i) in queue1, queue2, queue3 early than customer(i-1)
Yes
Set the timeSvcBegins(i) = timeSvcEnds(i-1)
Set the timeSvcEnds(i) = timeSvcBgins(i) + custServ(i)
No
i++
Compute waiting time and assign value to waitingTime
Compute spend time and assign value to timeSpend
Display customer departure, service of customer begins message
Call function printCounterResults to display detailed results of each customer
Call function evalResults
END

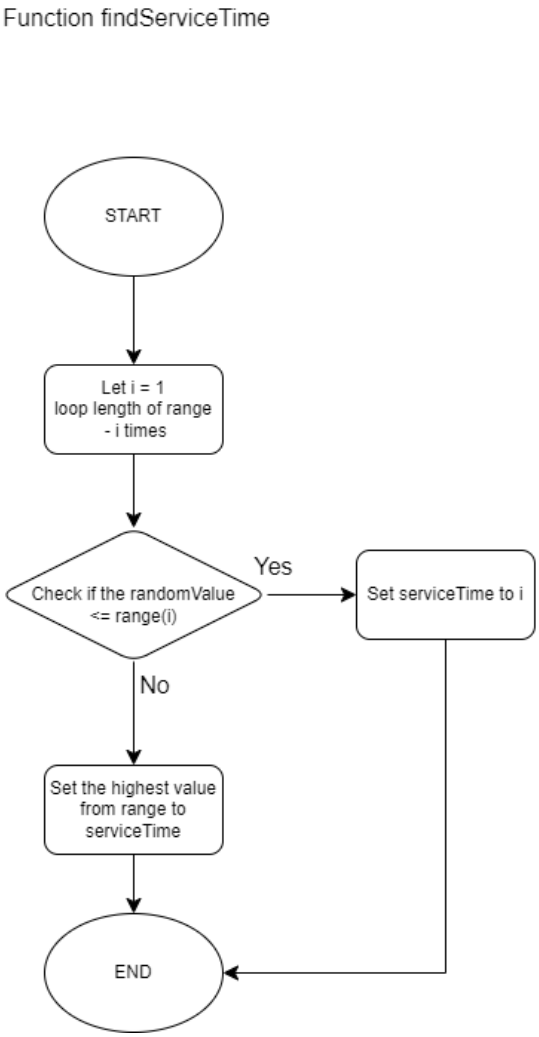
Supporting function in main:

a) `findServiceTime` function:

```
function serviceTime = findServiceTime(randomValue, range)
    % Determine service time based on ranges
    for i = 1:length(range)
        if randomValue <= range(i)
            serviceTime = i;
            return;
        end
    end
    serviceTime = length(range); % fallback to the highest value
end
```

In the `main` function, after generating the random service times and ranges for each counter, the `findServiceTime` function is called to determine the actual service time for each customer based on their random value.

Flowchart:



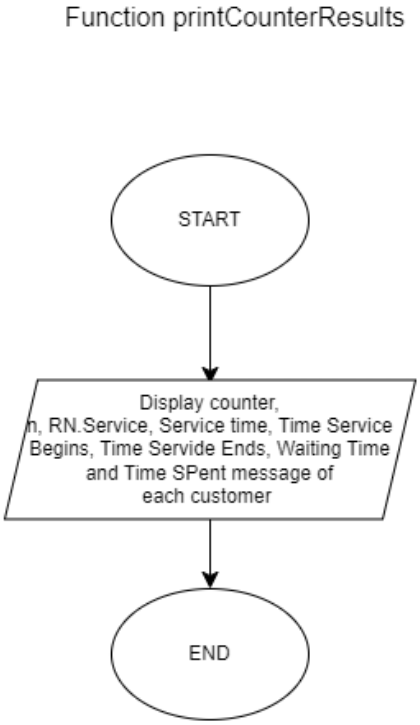
b) `printCounterResults` function:

```
function printCounterResults(counterName, queue, svcTime, custServ, timeSvcBegins, timeSvcEnds, waitingTime, timeSpend)
    fprintf('\n%s\n', counterName);
    fprintf('| n | RN.Service | Service time | Time Service Begins | Time Service Ends | Waiting Time | Time Spent |\n');
    for i = 1:numel(queue)
        customerNum = queue(i);
        fprintf('| %3d | %5d | %4d | %6d | %6d | %6d | %6d |\n', ...
            customerNum, svcTime(customerNum), custServ(customerNum), ...
            timeSvcBegins(customerNum), timeSvcEnds(customerNum), ...
            waitingTime(customerNum), timeSpend(customerNum));
    end
end
```

The `printCounterResults` function is designed to print detailed results for each counter in the simulation. It is included in the `main` function to provide an organized and structured way to output the results of the queuing simulation.

In summary, the `main` function models a car wash system, tracking customer arrivals, service times, queue assignments, and overall system performance.

Flowchart:



2.5 counter function:

The function calculates the cumulative distribution function (CDF) and ranges for service times, updates the `rangeCounter` matrix, and displays the details.

```
function rangeCounter=counter (loopNum, counterNum, probability, rangeCounter)
    %this function prints counter details

    %setting cdf
    cdf = zeros(1,6);
    cdf(1) = probability(1);
    for i=2:6
        cdf(i) = probability(i) + cdf(i-1);
    end

    %setting range
    range = zeros(1, 6);
    for i = 1:6
        range(i) = cdf(i)*100;
        rangeCounter(counterNum, i) = range(i);
    end

    rangeCounter = range;
```

The `cdf` is initialized as a zero array of size 6 (assumed to be the number of service times). The first element of `cdf` is set to the first probability. A loop then iterates from the second element to the sixth element, updating each `cdf` value to be the sum of the current probability and the previous `cdf` value.

The `range` is initialized as a zero array of size 6. A loop then iterates over the `cdf` values, multiplying each by 100 to get the range values, and updates the `rangeCounter` matrix for the specified counter.

```
    disp(' ');
    disp(['Counter ', num2str(counterNum), ':']);

    fprintf('Service Time|');
    for i=1:loopNum
        fprintf('   %d   |', i);
    end
    fprintf('\n');
```

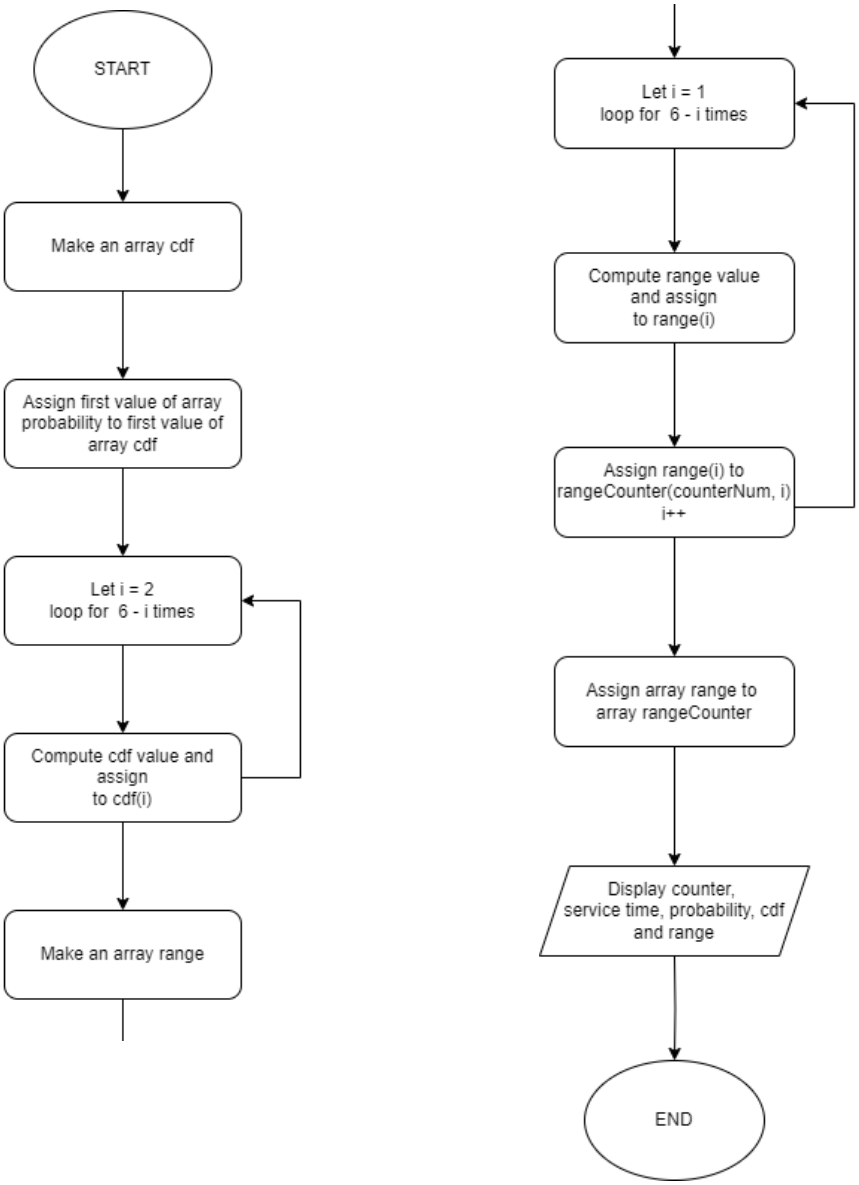
A blank line, counter number and header line for the service times is displayed.

```
%checking if arrays work
%disp(cdf);
%disp(range) ;

fprintf('Probability | ');
for i = 1:length(probability) %probability retrieved from main.m
    fprintf(' %.2f | ', probability(i));
end
fprintf('\n');

fprintf('CDF         | ');
for i = 1:length(cdf)
    fprintf(' %.2f | ', cdf(i));
end
fprintf('\n');

fprintf('Range       | ');
fprintf(' 0 -%2d |', range(1));
for i = 2:length(range)
    fprintf('%2d-%2d |', range(i-1) + 1, range(i));
end
fprintf('\n');
```

The probabilities and the CDF values are displayed in a formatted line. The ranges are displayed in a formatted line, showing the range intervals based on the `range` array.

The `counter` function calculates the CDF and ranges for service times based on input probabilities, updates the `rangeCounter` matrix, and displays the detailed results for each counter. This function helps to allocate service times based on predefined probabilities.

Flowchart:


START
Make an array cdf
Assign first value of array probability to first value of array cdf
Let i = 2
loop for 6 - i times
Compute cdf value and assign to cdf(i)
Make an array range
Let i = 1
loop for 6 - i times
Compute range value and assign to range(i)
Assign range(i) to rangeCounter(counterNum, i)
i++
Assign array range to array rangeCounter
Display counter, service time, probability, cdf and range
END

2.6 interTable function:

The function calculates probabilities, CDF, and ranges for inter-arrival times using different random number generators, normalizes these values, and updates the `rangeArrival` array. It also displays the inter-arrival probability table.

```
function rangeArrival = interTable(r, loopNum, m, a, c, rangeArrival)
    %this function prints inter arrival probability table

    loopNum = 4;

    if r == 1
    for i=1:loopNum
        probability(i) = rand();
        probability = probability / sum(probability); %normalising so that sum(probability) = 1
        probability = round(probability * 100) / 100;
    end

    elseif r == 2
        probability = lcg(loopNum,m,a,c);

    elseif r == 3
        probability = rvge(loopNum) ;

    elseif r == 4
        probability = rvgu(loopNum) ;

    else
        fprintf('zero \n');
        return;
    end
```

`loopNum` is set to 4, which means there will be 4 columns in the probability table.

- For `r == 1`, random numbers are generated using `rand()` and normalized.
- For `r == 2`, `lcg` generates the probabilities.
- For `r == 3`, `rvge` generates the probabilities.
- For `r == 4`, `rvgu` generates the probabilities.
- If `r` does not match any of these, the function prints `zero` and returns.

```
  % normalising so that sum(probability) = 1
  probability = probability / sum(probability);
  probability = round(probability*100)/100;

  % sometimes sum(probability) will not equal 1 because each value is rounded so here the leftover is
  if sum(probability) ~= 1
      leftover = 1 -sum(probability);
      index = randi(1,loopNum) ;
      probability(index) = probability(index) + leftover;
  end

  % returns probability
  output = probability;
```

The probabilities are normalized so their sum equals 1 and then rounded to two decimal places. If the sum of probabilities is not exactly 1 due to rounding, the leftover is added to a random element.

```
% setting cdf
cdf= zeros(1,4) ;
cdf(1) = probability(1);
for i=2:4
    cdf(i) =probability(i) + cdf(i-1);
end

% setting range
range = zeros(1,4) ;
for i = 1:4
    range(i) = cdf(i)*100;
    rangeArrival(i) = range(i);
end
```

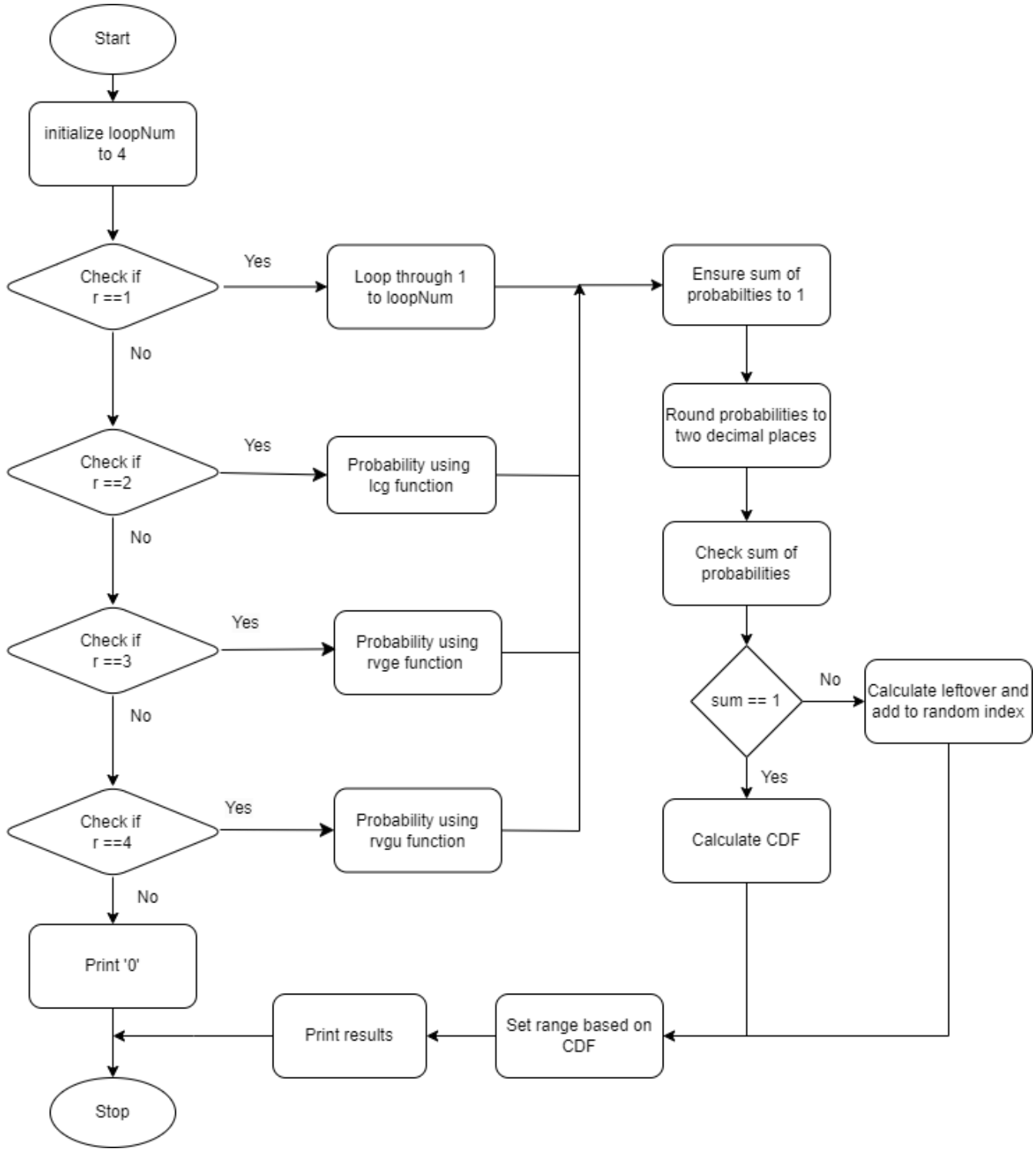
- `cdf` is calculated by accumulating the probabilities.
- `range` is calculated by multiplying `cdf` values by 100.
- `rangeArrival` is updated with the calculated ranges.

```
disp(' ');

fprintf('Arrival Time|');
for i=1:loopNum
    fprintf('   %d   |', i);
end
fprintf('\n');

% checking if arrays work
% disp(cdf);
% disp(range);

fprintf('Probability | ');
%probability retrieved from main.m
for i = 1:length(probability)
    fprintf('  %.2f  | ', probability(i));
end
fprintf('\n');

fprintf('CDF         | ');
for i = 1:length(cdf)
    fprintf (' %.2f  | ', cdf(i));
end
fprintf('\n');

fprintf('Range       | ');
fprintf('0 - %2d |', range(1)) ;
for i = 2:length(range)
    fprintf (' %2d -%2d|', range(i-1)+1, range(i));
end
fprintf('\n');
```

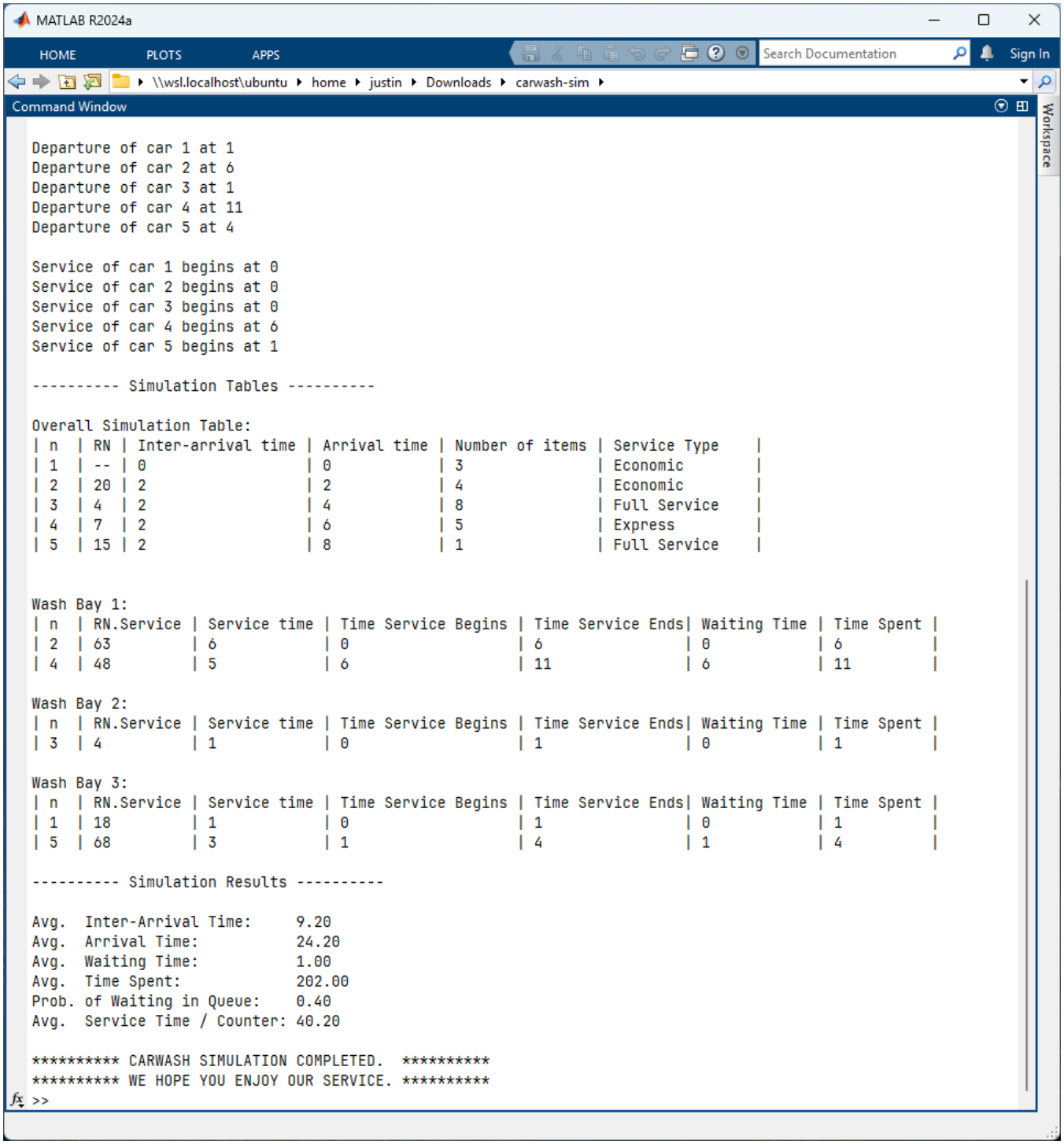
The function prints headers for the table. It then prints the probabilities, CDF values, and ranges in a formatted manner.

The `interTable` function computes and displays the inter-arrival probability table based on various random number generators. It normalizes the probabilities, calculates the CDF and ranges, updates the `rangeArrival` array, and displays the results in a formatted table. This function helps for simulations involving inter-arrival times of events

Flowchart:


Start
initialize loopNum to 4
Check if r ==1
Yes
Loop through 1 to loopNum
No
Check if r ==2
Yes
Probability using lcg function
No
Check if r ==3
Yes
Probability using rvge function
No
Check if r ==4
Yes
Probability using rvgu function
No
Print '0'
Ensure sum of probabilties to 1
Round probabilities to two decimal places
Check sum of probabilities
sum == 1
No
Calculate leftover and add to random index
Yes
Calculate CDF
Set range based on CDF
Print results
Stop

3. Simulation Tables and Results



MATLAB R2024a

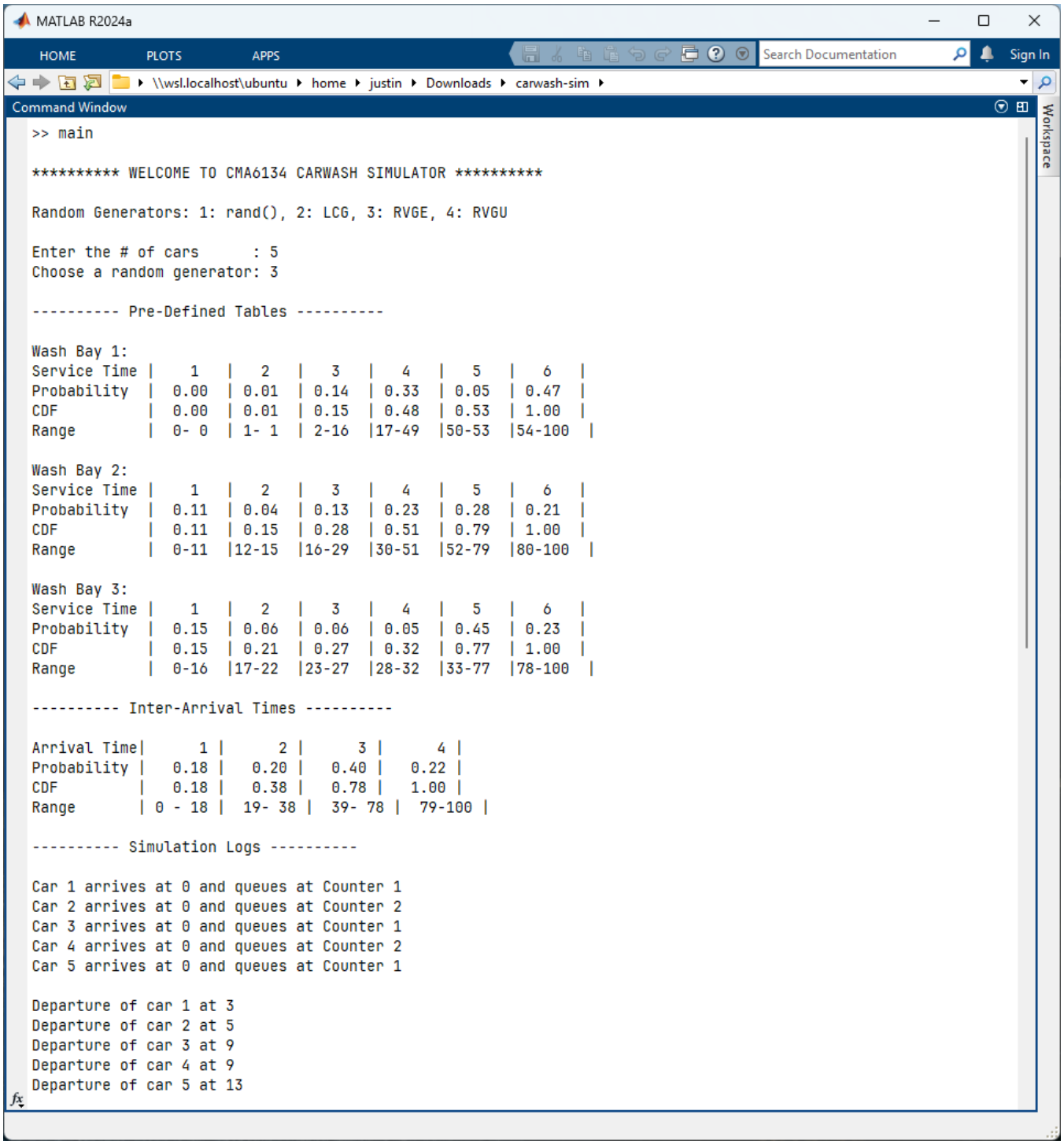
HOME PLOTS APPS Search Documentation Sign In

\\wsl.localhost\ubuntu ▸ home ▸ justin ▸ Downloads ▸ carwash-sim ▸

Command Window

```
>> main

********** WELCOME TO CMA6134 CARWASH SIMULATOR **********

Random Generators: 1: rand(), 2: LCG, 3: RVGE, 4: RVGU

Enter the # of cars      : 5
Choose a random generator: 4

---------- Pre-Defined Tables ----------

Wash Bay 1:
Service Time |    1    |    2    |    3    |    4    |    5    |    6    |
Probability  |  0.06   | 0.12    | 0.15    | 0.09    | 0.17    | 0.41    |
CDF          |  0.06   | 0.18    | 0.33    | 0.42    | 0.59    | 1.00    |
Range        |  0- 6   | 7-18    |19-33    |34-42    |43-59    |60-100   |

Wash Bay 2:
Service Time |    1    |    2    |    3    |    4    |    5    |    6    |
Probability  |  0.08   | 0.21    | 0.15    | 0.15    | 0.20    | 0.21    |
CDF          |  0.08   | 0.29    | 0.44    | 0.59    | 0.79    | 1.00    |
Range        |  0- 8   | 9-29    |30-44    |45-59    |60-79    |80-100   |

Wash Bay 3:
Service Time |    1    |    2    |    3    |    4    |    5    |    6    |
Probability  |  0.42   | 0.06    | 0.36    | 0.09    | 0.01    | 0.06    |
CDF          |  0.42   | 0.48    | 0.84    | 0.93    | 0.94    | 1.00    |
Range        |  0-42   |43-48    |49-84    |85-93    |94-94    |95-100   |

---------- Inter-Arrival Times ----------

Arrival Time|      1 |      2 |      3 |      4 |
Probability |   0.01 |   0.23 |   0.26 |   0.50 |
CDF         |   0.01 |   0.24 |   0.50 |   1.00 |
Range       | 0 -  1 |   2- 25 |  26- 50 |  51-100 |

---------- Simulation Logs ----------

Car 1 arrives at 0 and queues at Counter 3
Car 2 arrives at 0 and queues at Counter 1
Car 3 arrives at 0 and queues at Counter 2
Car 4 arrives at 0 and queues at Counter 1
Car 5 arrives at 0 and queues at Counter 3

Departure of car 1 at 1
Departure of car 2 at 6
Departure of car 3 at 1
Departure of car 4 at 11
Departure of car 5 at 4
```

Workspace


MATLAB R2024a
HOME
PLOTS
APPS
Search Documentation
Sign In
\\wsl.localhost\ubuntu ▸ home ▸ justin ▸ Downloads ▸ carwash-sim ▸
Command Window
Workspace
>> main
********** WELCOME TO CMA6134 CARWASH SIMULATOR **********
Random Generators: 1: rand(), 2: LCG, 3: RVGE, 4: RVGU
Enter the # of cars : 5
Choose a random generator: 3
---------- Pre-Defined Tables ----------
Wash Bay 1:
Service Time | 1 | 2 | 3 | 4 | 5 | 6 |
Probability | 0.00 | 0.01 | 0.14 | 0.33 | 0.05 | 0.47 |
CDF | 0.00 | 0.01 | 0.15 | 0.48 | 0.53 | 1.00 |
Range | 0- 0 | 1- 1 | 2-16 |17-49 |50-53 |54-100 |
Wash Bay 2:
Service Time | 1 | 2 | 3 | 4 | 5 | 6 |
Probability | 0.11 | 0.04 | 0.13 | 0.23 | 0.28 | 0.21 |
CDF | 0.11 | 0.15 | 0.28 | 0.51 | 0.79 | 1.00 |
Range | 0-11 |12-15 |16-29 |30-51 |52-79 |80-100 |
Wash Bay 3:
Service Time | 1 | 2 | 3 | 4 | 5 | 6 |
Probability | 0.15 | 0.06 | 0.06 | 0.05 | 0.45 | 0.23 |
CDF | 0.15 | 0.21 | 0.27 | 0.32 | 0.77 | 1.00 |
Range | 0-16 |17-22 |23-27 |28-32 |33-77 |78-100 |
---------- Inter-Arrival Times ----------
Arrival Time| 1 | 2 | 3 | 4 |
Probability | 0.18 | 0.20 | 0.40 | 0.22 |
CDF | 0.18 | 0.38 | 0.78 | 1.00 |
Range | 0 - 18 | 19- 38 | 39- 78 | 79-100 |
---------- Simulation Logs ----------
Car 1 arrives at 0 and queues at Counter 1
Car 2 arrives at 0 and queues at Counter 2
Car 3 arrives at 0 and queues at Counter 1
Car 4 arrives at 0 and queues at Counter 2
Car 5 arrives at 0 and queues at Counter 1
Departure of car 1 at 3
Departure of car 2 at 5
Departure of car 3 at 9
Departure of car 4 at 9
Departure of car 5 at 13

MATLAB R2024a

HOME PLOTS APPS Search Documentation Sign In

\\wsl.localhost\ubuntu ▸ home ▸ justin ▸ Downloads ▸ carwash-sim ▸

Command Window

```
Departure of car 1 at 3
Departure of car 2 at 5
Departure of car 3 at 9
Departure of car 4 at 9
Departure of car 5 at 13

Service of car 1 begins at 0
Service of car 2 begins at 0
Service of car 3 begins at 3
Service of car 4 begins at 5
Service of car 5 begins at 9

---------- Simulation Tables ----------

Overall Simulation Table:
| n  | RN | Inter-arrival time | Arrival time | Number of items | Service Type    |
| 1  | -- | 0                  | 0            | 10              | Full Service    |
| 2  | 95 | 4                  | 4            | 4               | Express         |
| 3  | 74 | 3                  | 7            | 4               | Economic        |
| 4  | 65 | 3                  | 10           | 9               | Express         |
| 5  | 19 | 2                  | 12           | 6               | Economic        |


Wash Bay 1:
| n  | RN.Service | Service time | Time Service Begins | Time Service Ends| Waiting Time | Time Spent |
| 1  | 8          | 3            | 0                   | 3                | 0            | 3          |
| 3  | 97         | 6            | 3                   | 9                | 3            | 9          |
| 5  | 27         | 4            | 9                   | 13               | 9            | 13         |

Wash Bay 2:
| n  | RN.Service | Service time | Time Service Begins | Time Service Ends| Waiting Time | Time Spent |
| 2  | 72         | 5            | 0                   | 5                | 0            | 5          |
| 4  | 32         | 4            | 5                   | 9                | 5            | 9          |

Wash Bay 3:
| n  | RN.Service | Service time | Time Service Begins | Time Service Ends| Waiting Time | Time Spent |

---------- Simulation Results ----------

Avg.  Inter-Arrival Time:     50.60
Avg.  Arrival Time:           150.20
Avg.  Waiting Time:           17.80
Avg.  Time Spent:             253.80
Prob. of Waiting in Queue:    0.40
Avg.  Service Time / Counter: 47.20

********** CARWASH SIMULATION COMPLETED.  **********
********** WE HOPE YOU ENJOY OUR SERVICE. **********
>>
```

MATLAB R2024a

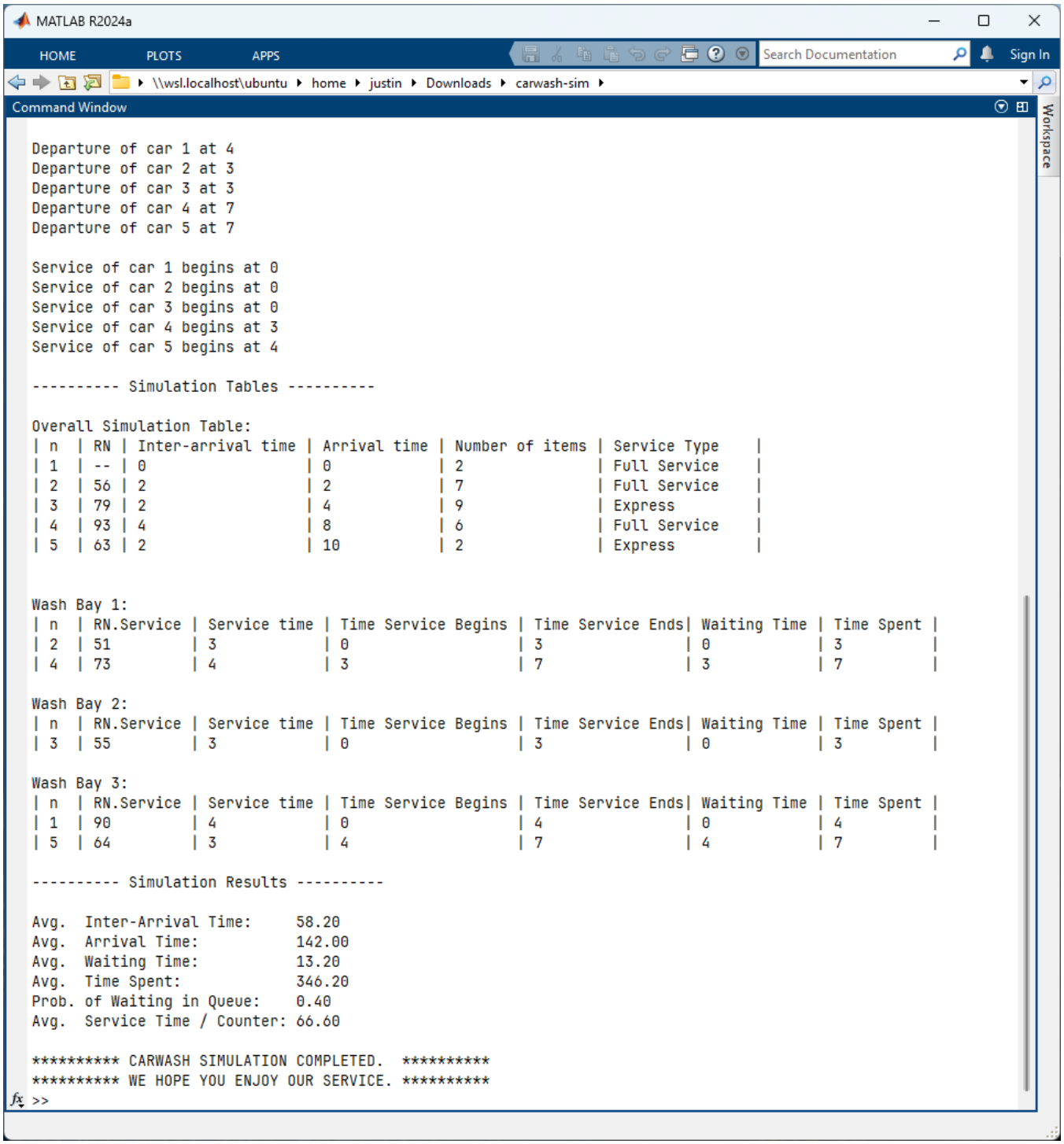
HOME PLOTS APPS Search Documentation Sign In

\\wsl.localhost\ubuntu ▸ home ▸ justin ▸ Downloads ▸ carwash-sim ▸

Command Window

```
>> main

********** WELCOME TO CMA6134 CARWASH SIMULATOR **********

Random Generators: 1: rand(), 2: LCG, 3: RVGE, 4: RVGU

Enter the # of cars      : 5
Choose a random generator: 2
Enter the modulus (m): 1
Enter the multiplier (a): 5
Enter the additive constant (c): 7

---------- Pre-Defined Tables ----------

Wash Bay 1:
Service Time |    1   |   2   |   3   |   4   |   5   |   6   |
Probability  |  0.25  | 0.23  | 0.14  | 0.21  | 0.07  | 0.10  |
CDF          |  0.25  | 0.48  | 0.62  | 0.83  | 0.90  | 1.00  |
Range        |  0-26  |27-49  |50-63  |64-83  |84-91  |92-101  |

Wash Bay 2:
Service Time |    1   |   2   |   3   |   4   |   5   |   6   |
Probability  |  0.28  | 0.16  | 0.21  | 0.15  | 0.13  | 0.07  |
CDF          |  0.28  | 0.44  | 0.65  | 0.80  | 0.93  | 1.00  |
Range        |  0-29  |30-45  |46-65  |66-80  |81-93  |94-100  |

Wash Bay 3:
Service Time |    1   |   2   |   3   |   4   |   5   |   6   |
Probability  |  0.26  | 0.12  | 0.29  | 0.25  | 0.06  | 0.02  |
CDF          |  0.26  | 0.38  | 0.67  | 0.92  | 0.98  | 1.00  |
Range        |  0-26  |27-38  |39-67  |68-92  |93-98  |99-100  |

---------- Inter-Arrival Times ----------

Arrival Time|      1 |      2 |      3 |      4 |
Probability |   0.46 |   0.33 |   0.12 |   0.09 |
CDF         |   0.46 |   0.79 |   0.91 |   1.00 |
Range       | 0 - 46 |  47- 79 |  80- 91 |  92-100 |

---------- Simulation Logs ----------

Car 1 arrives at 0 and queues at Counter 3
Car 2 arrives at 0 and queues at Counter 1
Car 3 arrives at 0 and queues at Counter 2
Car 4 arrives at 0 and queues at Counter 1
Car 5 arrives at 0 and queues at Counter 3

Departure of car 1 at 4
Departure of car 2 at 3
Departure of car 3 at 3
```

Workspace

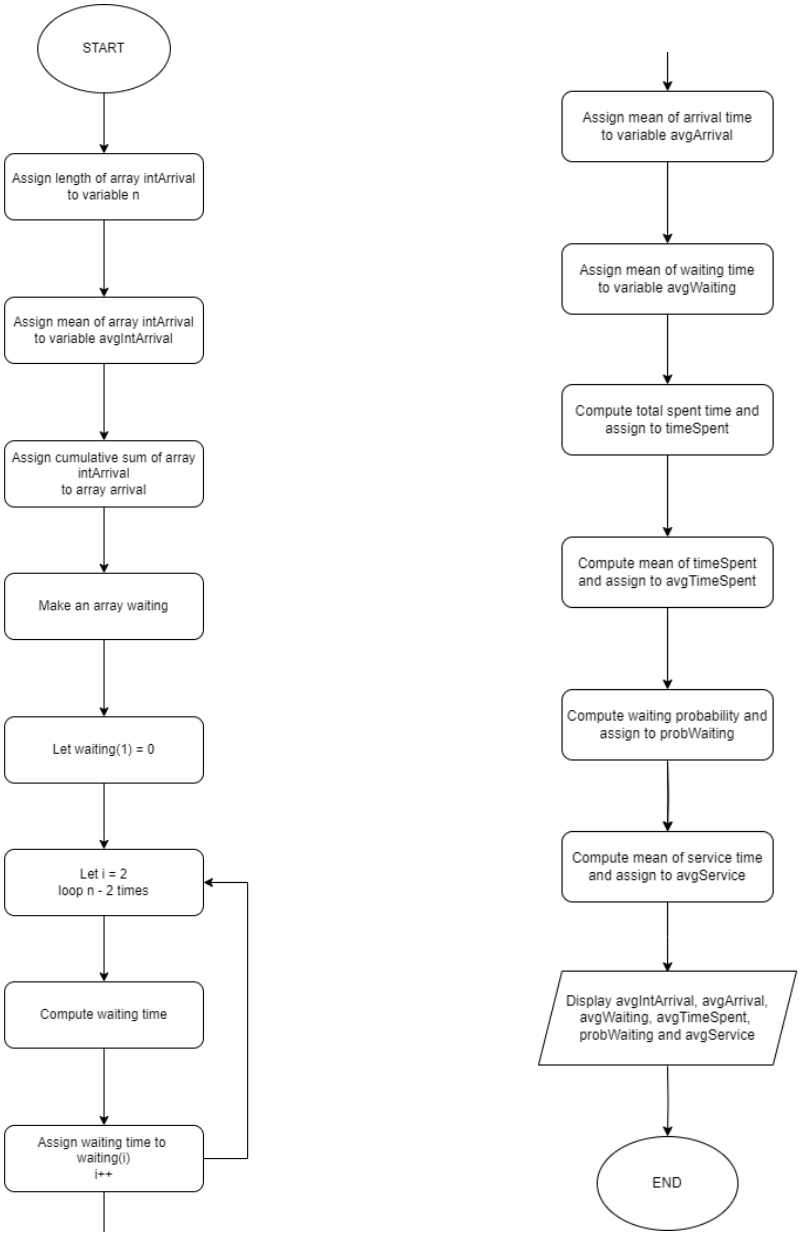

MATLAB R2024a
HOME
PLOTS
APPS
Search Documentation
Sign In
\\wsl.localhost\ubuntu ▸ home ▸ justin ▸ Downloads ▸ carwash-sim ▸
Command Window
Workspace
Departure of car 1 at 4
Departure of car 2 at 3
Departure of car 3 at 3
Departure of car 4 at 7
Departure of car 5 at 7
Service of car 1 begins at 0
Service of car 2 begins at 0
Service of car 3 begins at 0
Service of car 4 begins at 3
Service of car 5 begins at 4
---------- Simulation Tables ----------
Overall Simulation Table:
n	RN	Inter-arrival time	Arrival time	Number of items	Service Type
1	--	0	0	2	Full Service
2	56	2	2	7	Full Service
3	79	2	4	9	Express
4	93	4	8	6	Full Service
5	63	2	10	2	Express
Wash Bay 1:
n	RN.Service	Service time	Time Service Begins	Time Service Ends	Waiting Time	Time Spent
2	51	3	0	3	0	3
4	73	4	3	7	3	7
Wash Bay 2:
n	RN.Service	Service time	Time Service Begins	Time Service Ends	Waiting Time	Time Spent
3	55	3	0	3	0	3
Wash Bay 3:
n	RN.Service	Service time	Time Service Begins	Time Service Ends	Waiting Time	Time Spent
1	90	4	0	4	0	4
5	64	3	4	7	4	7
---------- Simulation Results ----------
Avg. Inter-Arrival Time: 58.20
Avg. Arrival Time: 142.00
Avg. Waiting Time: 13.20
Avg. Time Spent: 346.20
Prob. of Waiting in Queue: 0.40
Avg. Service Time / Counter: 66.60
********** CARWASH SIMULATION COMPLETED. **********
********** WE HOPE YOU ENJOY OUR SERVICE. **********
fx >>

3.5 Evaluation Results (evalResults function):

Code:

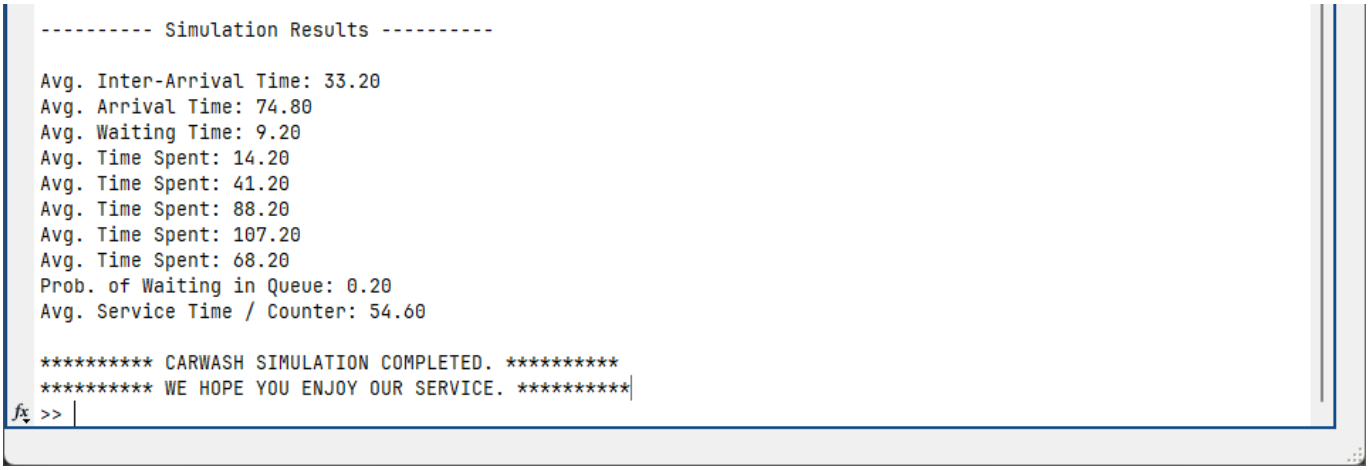
```
function evalResults(intArrival, service)
    % # of customers
    n = length(intArrival);

    avgIntArrival = mean(intArrival);
    arrival = cumsum(intArrival);

    % Waiting time for each customer
    % n rows, 1 column
    waiting = zeros(n, 1);
    waiting(1) = 0;
    for i = 2:n
        waiting(i) = max(0, arrival(i) - (arrival(i-1) + service(i-1)));
    end

    avgArrival = mean(arrival);
    avgWaiting = mean(waiting);
    timeSpent = waiting + service;
    avgTimeSpent = mean(timeSpent);
    probWaiting = sum(waiting > 0) / n;
    avgService = mean(service);


    fprintf('Simulation Results:\n');
    fprintf('Avg. Inter-Arrival Time: %.2f\n', avgIntArrival);
    fprintf('Avg. Arrival Time: %.2f\n', avgArrival);
    fprintf('Avg. Waiting Time: %.2f\n', avgWaiting);
    fprintf('Avg. Time Spent: %.2f\n', avgTimeSpent);
    fprintf('Prob. of Waiting in Queue: %.2f\n', probWaiting);
    fprintf('Avg. Service Time / Counter: %.2f\n', avgService);

% Average inter-arrival time
% Average arrival time
% Average waiting time
% Average time spent in the system
% Probability of waiting in the queue
% Average Service Time
```

Flowchart:


START
Assign length of array intArrival to variable n
Assign mean of array intArrival to variable avgIntArrival
Assign cumulative sum of array intArrival to array arrival
Make an array waiting
Let waiting(1) = 0
Let i = 2
loop n - 2 times
Compute waiting time
Assign waiting time to waiting(i)
i++
Assign mean of arrival time to variable avgArrival
Assign mean of waiting time to variable avgWaiting
Compute total spent time and assign to timeSpent
Compute mean of timeSpent and assign to avgTimeSpent
Compute waiting probability and assign to probWaiting
Compute mean of service time and assign to avgService
Display avgIntArrival, avgArrival, avgWaiting, avgTimeSpent, probWaiting and avgService
END

Screenshot result:

example



4. Conclusion

This simulation effectively models a car wash centre with three wash bays, providing insights into car arrivals, waiting times, and service times. The evaluation results, such as average waiting time and probability of waiting, help in understanding the efficiency of the car wash centre. This project demonstrates the use of random number generators and cumulative distribution functions in simulating real-world queuing systems.