MULTIMEDIA UNIVERSITY

TMA1301 COMPUTATIONAL METHODS

Trimester 2, 2022/2023

CODING ASSIGNMENT (20%)

Lecture Section: TC3L

Tutorial Section: T10L

Group Number: 1

Group Member Details:

No.	STUDENT NAME	ID NUMBER
1.	Mohd Farhan Bin Mohd Fairusham	1221303400
2.	Nur Arisya binti Mohd Yasak	1221303288
3.	Nur Farah Nabila binti Ramzairi	1221301140
4.	Sabrina Amalyn Binti Aminur Rizal	1201102251
5.	Wong Wai Yee	1221303660

Submission Date : Week 13, 16th June 2023

Table of Contents

1. Introduction

This report focuses on the development of a queue simulator specifically designed to replicate the arrival of customers at checkout counters in a supermarket. Our group’s objective is to create an extensive simulation system that takes various factors into account, such as the number of items acquired by each customer and their queue preferences. The primary goal is to analyse and evaluate the performance and efficiency of the supermarket’s checkout process using the simulation.

2. Simulation Details

2.1 Flow Charts

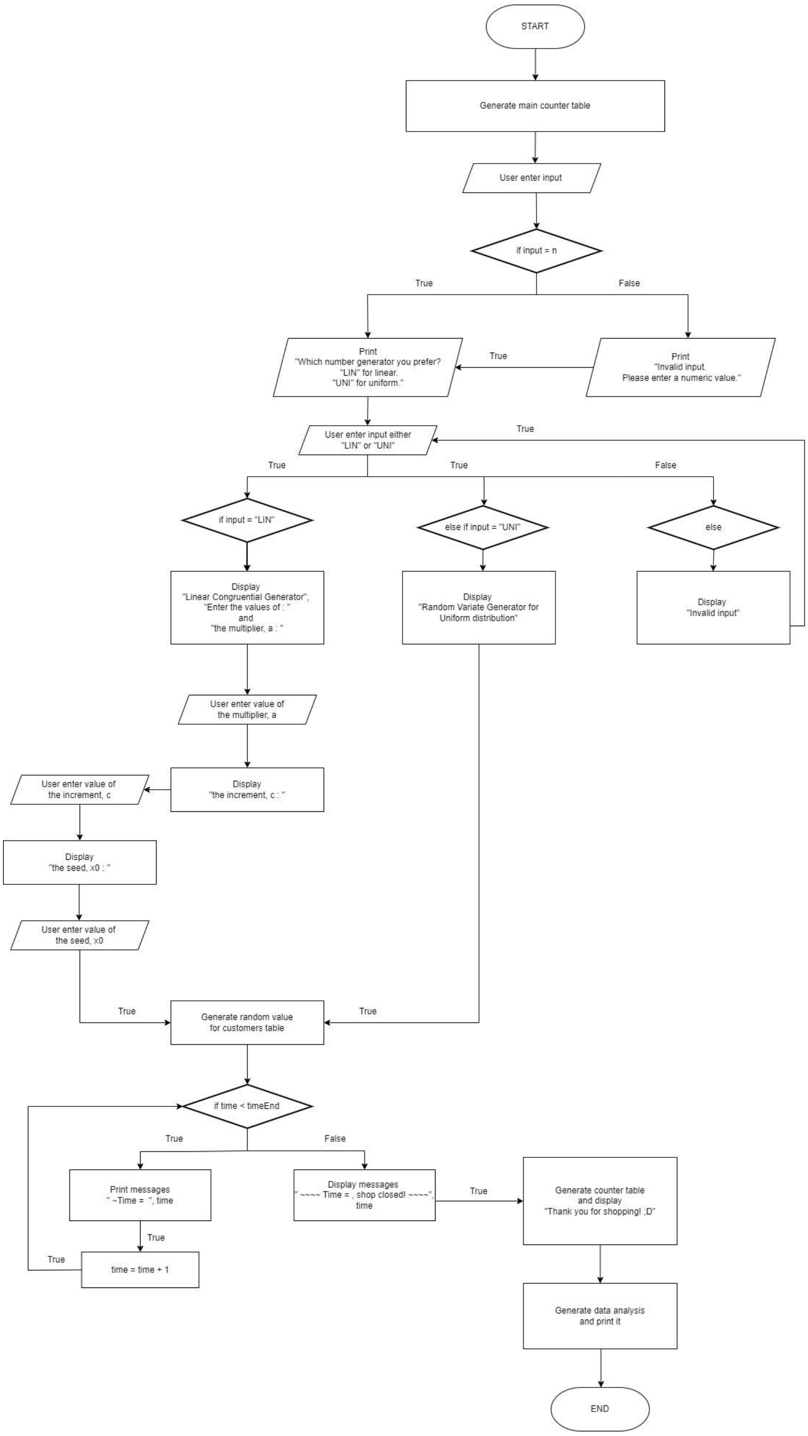


Figure 1: Flowchart for Entire Program

2.2 Implementation of Source Codes

2.2.1 main.m

```
function main()
    mainCounter();    % Creates the probabilities of counters
    mainCustomer();   % Creates the probabilities of customers
    mainGenRand();    % Creates the random numbers for customers
    mainWork();       % Assigns customers to counters
    analysis();       % Performs data analysis
```

Figure 2: main.m

The main() function serves as the central function that coordinates the execution of other functions within the program. It breaks down the program's logic into smaller modular pieces which helps improve readability, enhances program organisation, and makes it easier to test and debug.

2.2.2 mainCounter.m

```
function mainCounter()
    % Global variables
    counterAttributes = {'Service Time', 'Probability ', 'CDF ', 'Range '};

    global counter1Prob counter2Prob counterExProb;
    counterProbs = {counter1Prob, counter2Prob, counterExProb};

    global counterNum counterNames;
    counterNum = numel(counterProbs);
    counterNames = {'Counter 1  ', 'Counter 2  ', 'Counter Ex '};
    counterAttrs = {'Service Time', 'Probability ', 'CDF         ', 'Range
'};

    % Service Time
    global counterTimes;
    counter1Time = [5, 6, 7, 8, 9];
    counter2Time = [5, 6, 7, 8, 9];
    counterExTime = [2, 3, 4, 5];
    counterTimes = {counter1Time, counter2Time, counterExTime};

    columnNum1 = numel(counterTimes{1});
    columnNum2 = numel(counterTimes{2});
    columnNum3 = numel(counterTimes{3});
    columnNums = {columnNum1, columnNum2, columnNum3};
    % counterProbInit(); % For Testing

    for i = 1:counterNum
        counterProbs{i} = randArrayMaker(0.1, columnNums{i});
    end

    % Probabilities pakai Global

    % CDF
    counterCDF1 = 1:columnNums{1};
    counterCDF2 = 1:columnNums{2};
    counterCDF3 = 1:columnNums{3};
    counterCDFs = {counterCDF1, counterCDF2, counterCDF3};
    for i=1 : counterNum
        for j=1 : columnNums{i}
            if j == 1
                counterCDFs{i}(j) = counterProbs{i}(j);
            else
                counterCDFs{i}(j) = counterCDFs{i}(j-1) + counterProbs{i}(j);
            end
        end
    end

    % Range
    global counterRanges;
    counterRange1 = zeros(2, columnNums{1});
    counterRange2 = zeros(2, columnNums{2});
    counterRange3 = zeros(2, columnNums{3});
    counterRanges = {counterRange1, counterRange2, counterRange3};

    for i=1 : counterNum
        for j=1 : columnNums{i}
            switch j
                case 1
                    counterRanges{i}(1, j) = round(counterCDFs{i}(j)*100);
                    counterRanges{i}(2, j) = 0;
                otherwise
                    counterRanges{i}(1, j) = round(counterCDFs{i}(j)*100);
  % 1 is UB
                    counterRanges{i}(2, j) = round(counterCDFs{i}(j-1)*100 + 1);
  % 2 is LB
            end
        end
    end


    % Tables
    global counterArray;
    counterArray = {counterTimes, counterProbs, counterCDFs, counterRanges};
    for i=1 : counterNum

        fprintf('\n\n%s: \n', counterNames{i});
        for j=1 : numel(counterAttrs)
            switch j
                case 1
                    arrayStr = sprintf('   %d   |', counterArray{j}{i}(1,:));
                    disp([counterAttrs{j}, ': ', arrayStr]);
                case {2,3}
                    arrayStr = sprintf(' %.2f  |', counterArray{j}{i}(1,:));
                    disp([counterAttrs{j}, ': ', arrayStr]);
                case 4
                    lowerBounds = counterArray{j}{i}(2, :);
                    upperBounds = counterArray{j}{i}(1, :);
                    arrayStr = sprintf(' %02d-%d |', [lowerBounds;
upperBounds]);
                    disp([counterAttrs{j}, ': ', arrayStr]);
                otherwise
                    disp('Switch Error');
            end
        end
    end
    fprintf('\n');
```

Figure 3: mainCounter.m

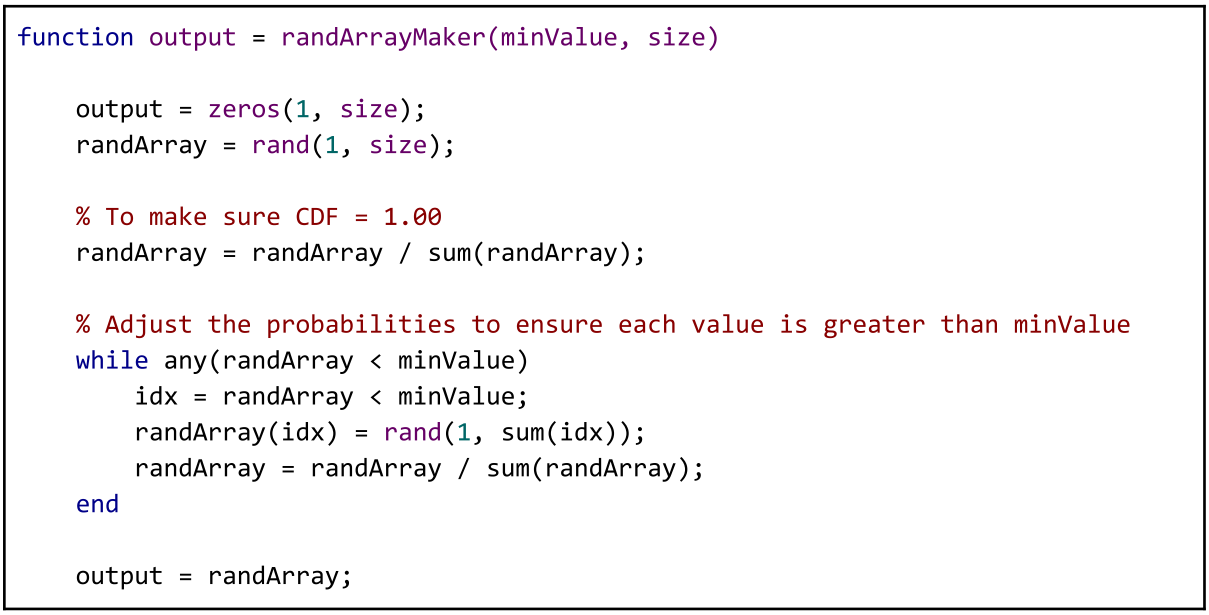
The mainCounter() function initialises and processes the service times, probabilities, cumulative distribution functions (CDFs), and ranges associated with counters in the program.

The mainCounter() function accomplishes its tasks by first setting up global variables and attributes specific to the counters. It then initialises the service times for the counters by declaring the global variable “counterTimes” and assigns specific service time values to each counter. These service time values are stored in the “counterTimes” variable, which is a cell array consisting of the service time arrays for each counter.

- counter1Time = [5, 6, 7, 8, 9]
- counter2Time = [5, 6, 7, 8, 9]
- counterExTime = [2, 3, 4, 5]

Next, it calculates the probabilities, CDFs, and ranges based on the generated data, using the randArrayMaker() function to generate probabilities. To compute the CDFs, it iterates through the probabilities and accumulates their values. These resulting CDFs determine the ranges, representing upper and lower bounds. Finally, the function stores all the counter-related data, including service times, probabilities, CDFs, and ranges, in a global variable named “counterArray”. It also displays this information in a table for easy comprehension and analysis.

2.2.2.1 randArrayMaker.m



```
function output = randArrayMaker(minValue, size)

    output = zeros(1, size);
    randArray = rand(1, size);

    % To make sure CDF = 1.00
    randArray = randArray / sum(randArray);

    % Adjust the probabilities to ensure each value is greater than minValue
    while any(randArray < minValue)
        idx = randArray < minValue;
        randArray(idx) = rand(1, sum(idx));
        randArray = randArray / sum(randArray);
    end

    output = randArray;
```

Figure 4: randArrayMaker.m

The randArrayMaker() function generates a random array of a specified size with adjusted probabilities.

The function accomplishes its tasks by initialising the output array and generating a random array. Then, it ensures that the cumulative distribution function (CDF) of the random array is equal to 1.0 by dividing each element by the sum of all elements. The function then adjusts the probabilities to make sure each value is greater than or equal to a specified minimum value. It iteratively generates new random values for elements that do not meet the minimum value condition and recalculates the CDF until all values satisfy the requirement. Finally, the adjusted random array is assigned to the output variable and returned. Overall, the randArrayMaker() function generates a random array with adjusted probabilities, ensuring the CDF is 1.0 and each element meets the minimum value criterion.

2.2.3 mainCustomer.m

```
function mainCustomer()
    custAttr = {'Arrival Time', 'Probability ', 'CDF         ', 'Range       '};

    global interArrTime;
    interArrTime = [1, 2, 3, 4];
    columnNum = numel(interArrTime);

    % Probability
    global custProb;
    % custProb = [0.2, 0.2, 0.2, 0.2, 0.2]; % For Testing
    custProb = randArrayMaker(0.1, columnNum);

    % CDF
    custCDF = zeros(1, columnNum);
    custCDF(1) = custProb(1);
    for i=2 : columnNum
        custCDF(i) = custCDF(i-1) + custProb(i);
    end

    % Range
    global custRange;
    custRange = zeros(2, columnNum);

    for i=1 : columnNum
        switch i
            case 1
                custRange(1, i) = round(custCDF(i)*100);
                custRange(2, i) = 0;
            otherwise
                custRange(1, i) = round(custCDF(i)*100);       % 1 is UB
                custRange(2, i) = round(custCDF(i-1)*100 + 1); % 2 is LB
        end
    end

    % Tables
    arrayOfArrays = {interArrTime, custProb, custCDF, custRange};

    fprintf('\nCustomer: \n');
    for i=1 : numel(custAttr)
```

```
        switch i
            case 1
                arrayStr = sprintf('   %d   |', arrayOfArrays{i});
                disp([custAttr{i}, ': ', arrayStr]);
            case {2,3}
                arrayStr = sprintf(' %.2f  |', arrayOfArrays{i});
                disp([custAttr{i}, ': ', arrayStr]);
            case 4
                lowerBounds = arrayOfArrays{i}(2, :);
                upperBounds = arrayOfArrays{i}(1, :);
                arrayStr = sprintf(' %02d-%d |', [lowerBounds; upperBounds]);
                disp([custAttr{i}, ': ', arrayStr]);
            otherwise
                disp('switch Error');
        end
    end
    fprintf('\n');
```

Figure 5: mainCustomer.m

The mainCustomer() function handles the initialisation, processing, and display of customer arrival time information within the program.

The function accomplishes its tasks by initialising the inter-arrival times and generates probabilities using the randArrayMaker() function. The function then calculates the cumulative distribution function (CDF) and determines the ranges for the arrival times. The data, including inter-arrival times, probabilities, CDFs, and ranges, is stored in an array. Finally, the function displays the customer-related data in a table.

2.2.4 mainGenRand.m

```
function mainGenRand()
    global randomNumbers;
    disp('Enter the number of customers:');
    custAmount = inputCheck();

    % myPause();

    answering = true;
    while answering
        fprintf('Which number generator you prefer? "LIN" for linear. "UNI" for 
uniform.\n');
        genType = input('LIN or UNI? -> ', 's');
        switch lower(genType)
            case 'lin'
                disp('Linear Congruential Generator');
                disp('Enter the values of:');
                disp('the multiplier, a: ');
                a = inputCheck();
                disp('the increment, c: ');
                c = inputCheck();
                disp('the seed, x0: ');
                seed = inputCheck();

                randomNumbers = genLCG(a, c, custAmount, seed);
                answering = false;

            case 'uni'
                disp('Random Variate Generator for Uniform distribution');
                randomNumbers = genRVGUD(custAmount);
                answering = false;

            otherwise
                disp('Invalid input.');
                myPause()
        end
    end

    % Inter-Arrival Time
    global custRange;
    for i = 1:custAmount
        interValues(i) = valueAssigner(randomNumbers(i), custRange);
    end

    % Arrival Time
    arrivalTimes(1) = 0;
    for i = 2:custAmount
        arrivalTimes(i) = arrivalTimes(i-1) + interValues(i);
    end

    % Items
    minValue = 1;
    maxValue = 10;
    % Generate random integers between minValue and maxValue
    items = floor(rand(1, custAmount) * maxValue + minValue);

    myPause();

    randAttrs = {'n ', 'RN', 'Inter', 'A.Time', 'Items'};
    fprintf('Random values for customers:\n');
    for i = 1:numel(randAttrs)
        if i < numel(randAttrs)
            fprintf(' %s |', randAttrs{i});
        else
            fprintf(' %s\n', randAttrs{i});
        end
    end
    fprintf('----+----+-------+--------+-------\n');
    for n = 1:custAmount
        fprintf(' %02d | %02d |   %d   |   %02d   |   %d \n', ...
        [n; randomNumbers(n); interValues(n); arrivalTimes(n); items(n)]);
    end


    % The customer Table
    global customers;
    customers = {1:custAmount, randomNumbers, interValues, arrivalTimes, items};
```

Figure 6: mainGenRand.m

The mainGenRand() function allows the user to generate random numbers for a specified number of customers.

The function accomplishes its tasks by first offering options for using a linear congruential generator (LCG) or a random variate generator for a uniform distribution. The function then assigns inter-arrival times and arrival times based on the generated random numbers, as well as generates random item values. It presents the resulting data in a table, providing information such as customer numbers, random numbers, inter-arrival times, arrival times, and item values.

2.2.4.1 myPause.m

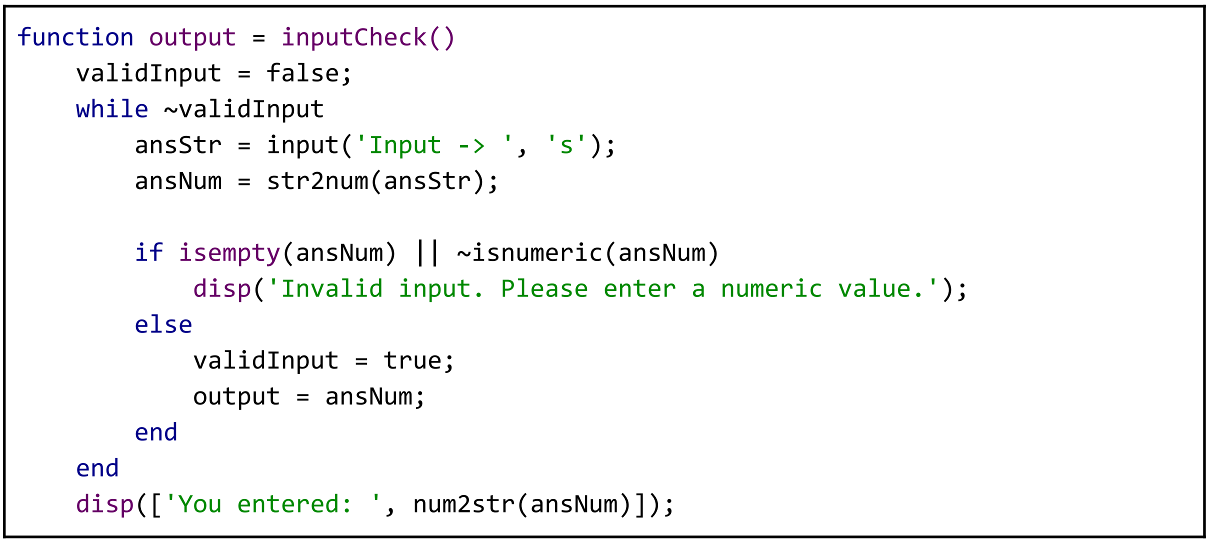
```
function output = myPause()
    fprintf('\n');
    output = getline('Press ENTER to continue...');
    fprintf('\n');
```

Figure 7: myPause.m

The myPause() function's purpose is to pause the program execution and prompt the user to press the ENTER key to continue.

Note: This function will be utilised in the other programs below, however its description will not be included to avoid unnecessary repetition.

2.2.4.2 inputCheck.m



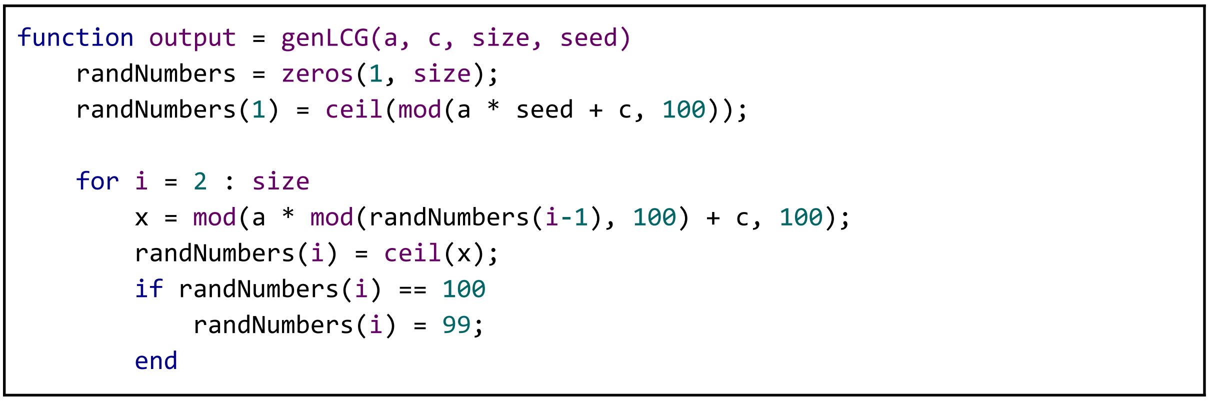
```
function output = inputCheck()
    validInput = false;
    while ~validInput
        ansStr = input('Input -> ', 's');
        ansNum = str2num(ansStr);

        if isempty(ansNum) || ~isnumeric(ansNum)
            disp('Invalid input. Please enter a numeric value.');
        else
            validInput = true;
            output = ansNum;
        end
    end
    disp(['You entered: ', num2str(ansNum)]);
```

Figure 8: inputCheck.m

The inputCheck() function validates and processes user input by repeatedly prompting the user to enter a numeric value until valid input is received.

The function accomplishes its tasks by checking if the input is numeric and not empty, displaying an error message if the input is invalid. Once valid input is provided, the function then exits the loop and stores the input value in the 'output' variable. It also provides feedback to the user by displaying the entered value.

2.2.4.3 genLCG.m



```
function output = genLCG(a, c, size, seed)
    randNumbers = zeros(1, size);
    randNumbers(1) = ceil(mod(a * seed + c, 100));

    for i = 2 : size
        x = mod(a * mod(randNumbers(i-1), 100) + c, 100);
        randNumbers(i) = ceil(x);
        if randNumbers(i) == 100
            randNumbers(i) = 99;
        end
    end

    output = randNumbers;
end
```

Figure 9: genLCG.m

The genLCG() function generates random numbers using a linear congruential generator algorithm.

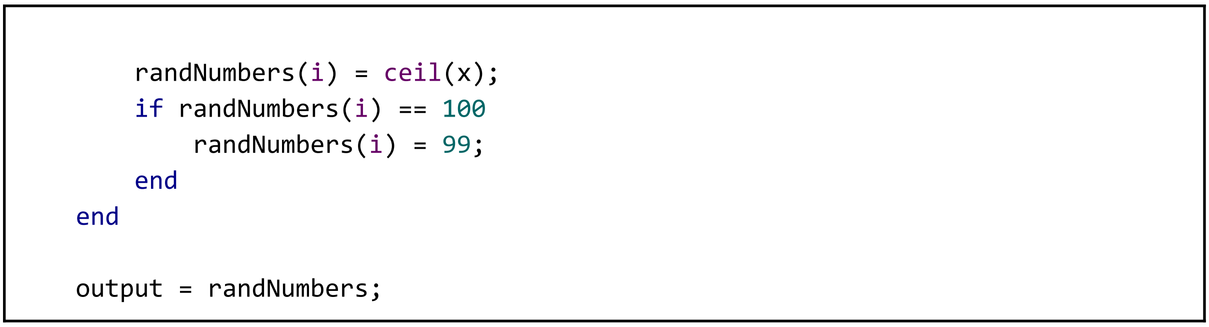
It takes four parameters:

1) **a**: multiplier
2) **c**: increment
3) **size**: number of random numbers to generate
4) **seed**: initial seed value

The function accomplishes its tasks by initialising an array “randNumbers” with the specified size. It calculates the first random number by applying the LCG formula to the seed. Then, it iteratively calculates subsequent random numbers by applying the formula to the previous number. The generated numbers are stored in the 'randNumbers' array, with each number rounded up to the nearest integer. If a generated number is equal to 100, it is adjusted to 99 to ensure it falls within the desired range. Finally, the “randNumbers” array is returned as the output of the function.

2.2.4.4 genRVGUD.m

```
function output = genRVGUD(size)
    randNumbers = zeros(1, size);
    a = 0;
    b = 100;

    for i=1 : size
        x = rand();
        x = a + (b - a)*x;
```



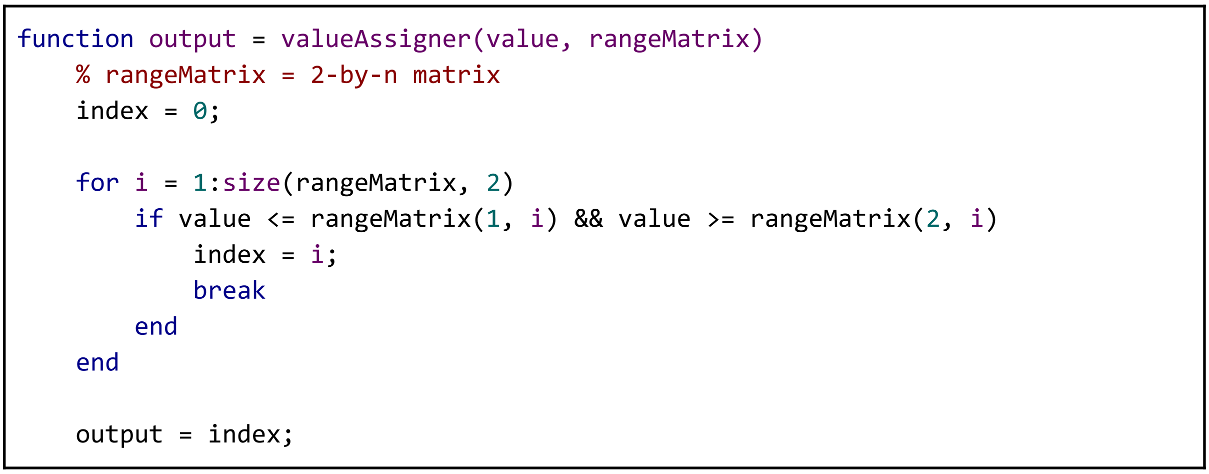
```
        randNumbers(i) = ceil(x);
        if randNumbers(i) == 100
            randNumbers(i) = 99;
        end
    end

    output = randNumbers;
```

Figure 10: genRVGUD.m

The genRVGUD() function generates random numbers from a uniform distribution.

The function accomplishes its tasks by taking one parameter, 'size', indicating the number of random numbers to generate. The function then initialises an array “randNumbers” with the specified size. It iterates “size” times and generates a random number “x” using the rand() function. “x” is then scaled to the desired range [a, b] using the formula a + (b - a)*x. The scaled number is rounded up to the nearest integer and stored in the 'randNumbers' array. If a generated number is equal to 100, it is adjusted to 99 to ensure it falls within the desired range. Finally, the 'randNumbers' array is returned as the output of the function.

2.2.4.5 valueAssigner.m



```
function output = valueAssigner(value, rangeMatrix)
    % rangeMatrix = 2-by-n matrix
    index = 0;

    for i = 1:size(rangeMatrix, 2)
        if value <= rangeMatrix(1, i) && value >= rangeMatrix(2, i)
            index = i;
            break
        end
    end

    output = index;
```

Figure 11: valueAssigner.m

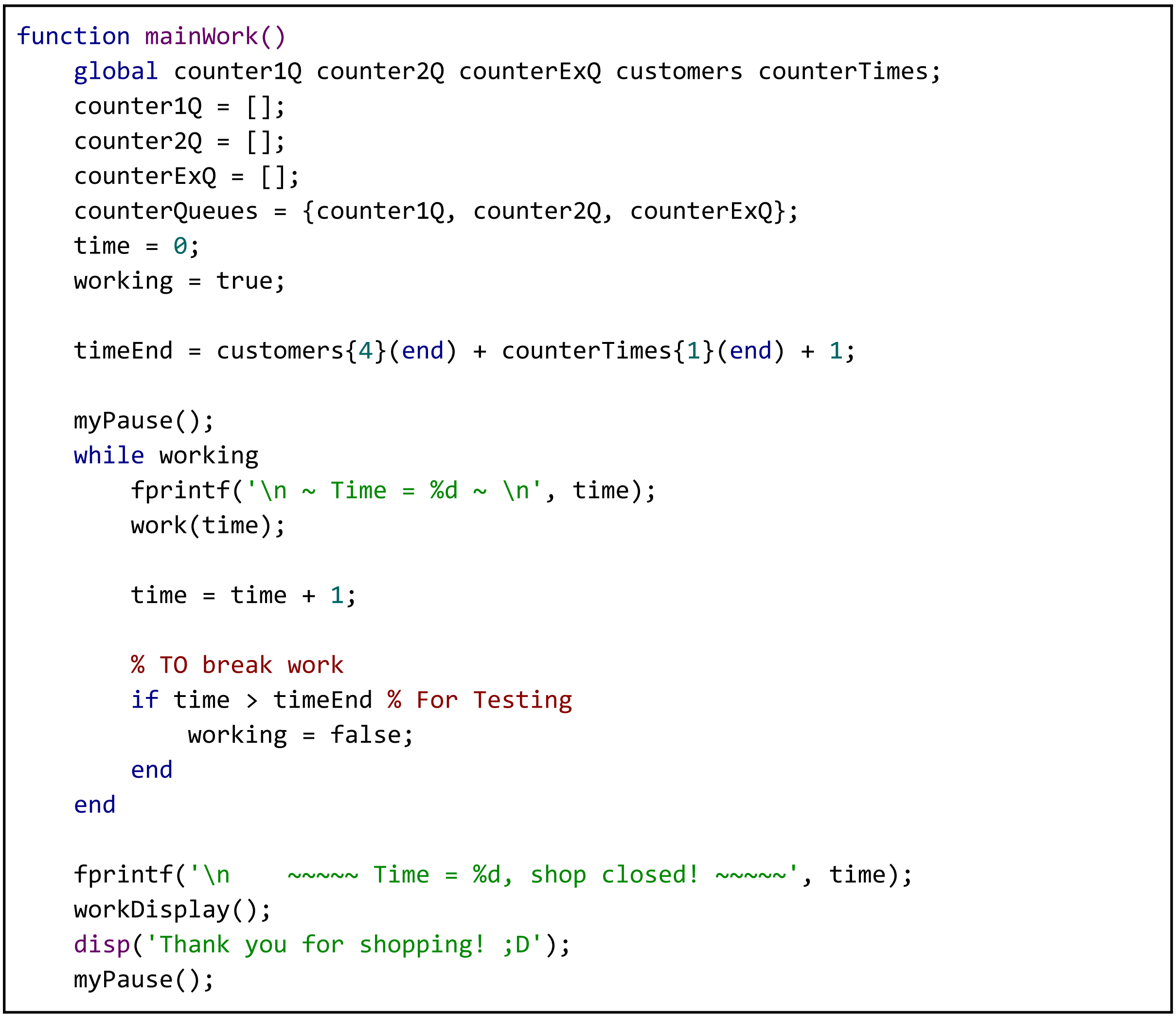
The valueAssigner() function takes two parameters:

1) **value:** represents the value to be assigned to a specific range.

2) **rangeMatrix:** a 2-by-n matrix containing the upper and lower bounds of the ranges.

The function accomplishes its tasks by initialising the variable "index" to 0. It then iterates through the columns of the "rangeMatrix" and checks if the "value" falls within the current range. If the "value" is greater than or equal to the lower bound and less than or equal to the upper bound of the range, the "index" is set to the current column number and the loop is terminated using the "break" statement. Finally, the "index" value is assigned to the "output" variable and returned by the function.

2.2.5 mainWork.m



```
function mainWork()
    global counter1Q counter2Q counterExQ customers counterTimes;
    counter1Q = [];
    counter2Q = [];
    counterExQ = [];
    counterQueues = {counter1Q, counter2Q, counterExQ};
    time = 0;
    working = true;

    timeEnd = customers{4}(end) + counterTimes{1}(end) + 1;

    myPause();
    while working
        fprintf('\n ~ Time = %d ~ \n', time);
        work(time);

        time = time + 1;

        % TO break work
        if time > timeEnd % For Testing
            working = false;
        end
    end

    fprintf('\n    ~~~~~ Time = %d, shop closed! ~~~~~', time);
    workDisplay();
    disp('Thank you for shopping! ;D');
    myPause();
```

Figure 12: mainWork.m

The mainWork() function initialises the counters and customer queues, runs a loop to simulate time passing, and calls the work() function at each time interval to process customers and update the queues. The simulation continues until a specified time is reached. The function also displays the current time, state of the counter and queues, and a message thanking the customers is displayed.

2.2.5.1 work.m

```
function work(timeParam)
    % Time Arrive, customer queue
    % Time Service ends, customer leave

    global customers counter1Q counter2Q counterExQ counterArray counterNames;
    counterQueues = {counter1Q, counter2Q, counterExQ};
    custAmount = numel(customers{1});

    % Queueing at Counters
    for i = 1:custAmount
        if timeParam == customers{4}(i) % Arrival Time
            fprintf('Customer %d has arrived ', customers{1}(i));
            if customers{5}(i) <= 3
                counterExQ = [counterExQ, i];
                fprintf('and queued at Counter Express.\n');
            elseif numel(counter1Q) <= numel(counter2Q)
                counter1Q = [counter1Q, i];
                fprintf('and queued at Counter 1.\n');
            else
                counter2Q = [counter2Q, i];
                fprintf('and queued at Counter 2.\n');
            end
        end
    end

    for c = 1:numel(counterQueues)

        % Initializing attributes
        queueAmount = numel(counterQueues{c});
        if queueAmount <= 0
            queueAmount = 1;
        end
        ID = zeros(1, queueAmount);
        RN = zeros(1, queueAmount);
        interArrival = zeros(1, queueAmount);
        arrival = zeros(1, queueAmount);
        service = zeros(1, queueAmount);
        serviceBegin = zeros(1, queueAmount);
        serviceEnd = zeros(1, queueAmount);
        wait = zeros(1, queueAmount);
        spent = zeros(1, queueAmount);

        for n = 1:numel(counterQueues{c}) % For every customer at each counter
            ID(n) = counterQueues{c}(n);
            RN(n) = customers{2}(ID(n));

            interArrival(n) = customers{3}(ID(n));
            arrival(n) = customers{4}(ID(n));

            service(n) = counterArray{1}{c}(valueAssigner(RN(n),
counterArray{4}{c}));
            serviceEnd(n) = service(n) + arrival(n);
            serviceBegin(2:end) = serviceEnd(1:end-1);

            wait(n) = serviceBegin(n) - arrival(n);
            if wait(n) < 0 % No Negative time
                wait(n) = 0;
            end
            spent(n) = serviceEnd(n) - arrival(n);
        end
        workAttrs{c} = {ID, RN, interArrival, arrival, service, serviceBegin,
serviceEnd, wait, spent};
    end

    global workAttrArray;
    workAttrArray = {workAttrs{1}, workAttrs{2}, workAttrs{3}};

        % Show Departure
    for c = 1:numel(counterQueues)
        for i = 1:numel(counterQueues{c})
            if timeParam == workAttrArray{c}{7}(i)
                fprintf('Customer %d has left %s\n', workAttrArray{c}{1}(i),
counterNames{c});
            end
        end
    end
```

Figure 13: work.m

The work() function handles the customer arrivals, queuing at counters, and customer departures. When a customer arrives, they are assigned to a counter queue based on their item count. The function then calculates the inter-arrival time, arrival time, service time, and wait time for each customer in the queue. These attributes are then stored and used to determine when customers complete their service and leave the counter. The function also displays the customer flow and counter operations at each time interval.

2.2.5.2 workDisplay.m

```
function workDisplay()
    global customers counterNames counterArray;
    global counter1Q counter2Q counterExQ workAttrArray;
    counterQueues = {counter1Q, counter2Q, counterExQ};

    workAttrStr = {'n ', 'RN','Inter', 'A.Time', 'Service', 'Begins', 'Ends',
'Wait', 'Spent'};
    custAmount = numel(customers{1});

    for c = 1:numel(counterQueues)
        fprintf('\n\n%s: \n', counterNames{c});

        for i = 1:numel(workAttrStr)
                if i < numel(workAttrStr)
                fprintf(' %s |', workAttrStr{i});
            else
                fprintf(' %s\n', workAttrStr{i});
            end
        end

fprintf('----+----+-------+--------+---------+--------+------+------+-------\n')
;
        for n = 1:numel(counterQueues{c}) % Number of customers queueing
            fprintf(' %02d | %02d |   %d   |   %02d   |    %d    |   %02d   |
%02d  |  %02d  |  %d \n', ...
            [workAttrArray{c}{1}(n); workAttrArray{c}{2}(n);
workAttrArray{c}{3}(n); ...
            workAttrArray{c}{4}(n); workAttrArray{c}{5}(n);
workAttrArray{c}{6}(n); ...
            workAttrArray{c}{7}(n); workAttrArray{c}{8}(n);
workAttrArray{c}{9}(n)]);
        end
    end
```

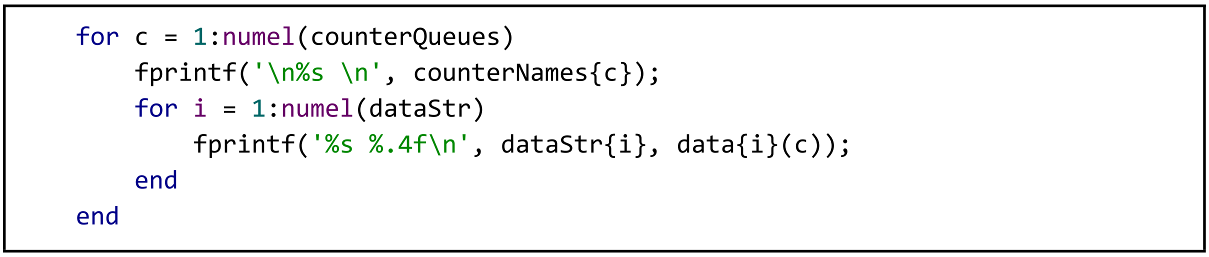
Figure 14: workDisplay.m

The workDisplay() function is responsible for displaying the attributes of customers at each counter queue during the work process.

The function accomplishes its tasks by first defining the “workAttrStr” variable, which contains the labels for the customer attributes. Then, for each counter queue, it displays the counter name and the attribute labels. It iterates over the customers in the queue and prints their corresponding attributes in a formatted manner using fprintf(). The attributes include the customer number, random number, inter-arrival time, arrival time, service time, service begin time, service end time, wait time, and time spent.

2.2.6 analysis.m

```
function analysis()
    global counter1Q counter2Q counterExQ customers;
    counterQueues = {counter1Q, counter2Q, counterExQ};
    % Data Analysis
    global data workAttrArray;

    % Initializing data
    interAvg = zeros(1, numel(counterQueues));
    arriveAvg = zeros(1, numel(counterQueues));
    serveAvg = zeros(1, numel(counterQueues));
    waitAvg = zeros(1, numel(counterQueues));
    spentAvg = zeros(1, numel(counterQueues));
    waitProb = zeros(1, numel(counterQueues));

    for c = 1:numel(counterQueues)
        interAvg(c) = mean(workAttrArray{c}{3});
        arriveAvg(c) = mean(workAttrArray{c}{4});
        serveAvg(c) = mean(workAttrArray{c}{5});
        waitAvg(c) = mean(workAttrArray{c}{8});
        spentAvg(c) = mean(workAttrArray{c}{9});

        count = sum(workAttrArray{c}{8} == 0);
        waitProb(c) = 1 - (count / numel(counterQueues{c}));
    end

    data = {interAvg, arriveAvg, serveAvg, waitAvg, spentAvg, waitProb};

    % Data Analysis Display
    global counterNames;

    dataStr = {'AVG. Inter-Arrival Time:', 'AVG. Arrival Time:', 'AVG. Service
Time:',...
    'AVG. Wait Time:', 'AVG. Spent Time:', 'Wait Probability:'};

    fprintf('Data Analysis\n');
```



```
    for c = 1:numel(counterQueues)
        fprintf('\n%s \n', counterNames{c});
        for i = 1:numel(dataStr)
            fprintf('%s %.4f\n', dataStr{i}, data{i}(c));
        end
    end
```

Figure 15: analysis.m

The analysis() function is responsible for performing data analysis on the attributes of customers at each counter queue.

The function accomplishes its tasks by first initialising variables to store the average values of inter-arrival time, arrival time, service time, wait time, time spent, and wait probability for each counter queue. It calculates these averages using the mean() function applied to the corresponding attributes in the workAttrArray. Additionally, it determines the wait probability by counting the number of customers with a wait time of 0 and dividing it by the total number of customers in the queue. The results are stored in the data variable. The function then displays the data analysis results, including the counter names and the average values of the analysed attributes for each counter queue.

2.3 Summary of All Functions

Table 1: Summary of all the Functions for the Coding Assignment

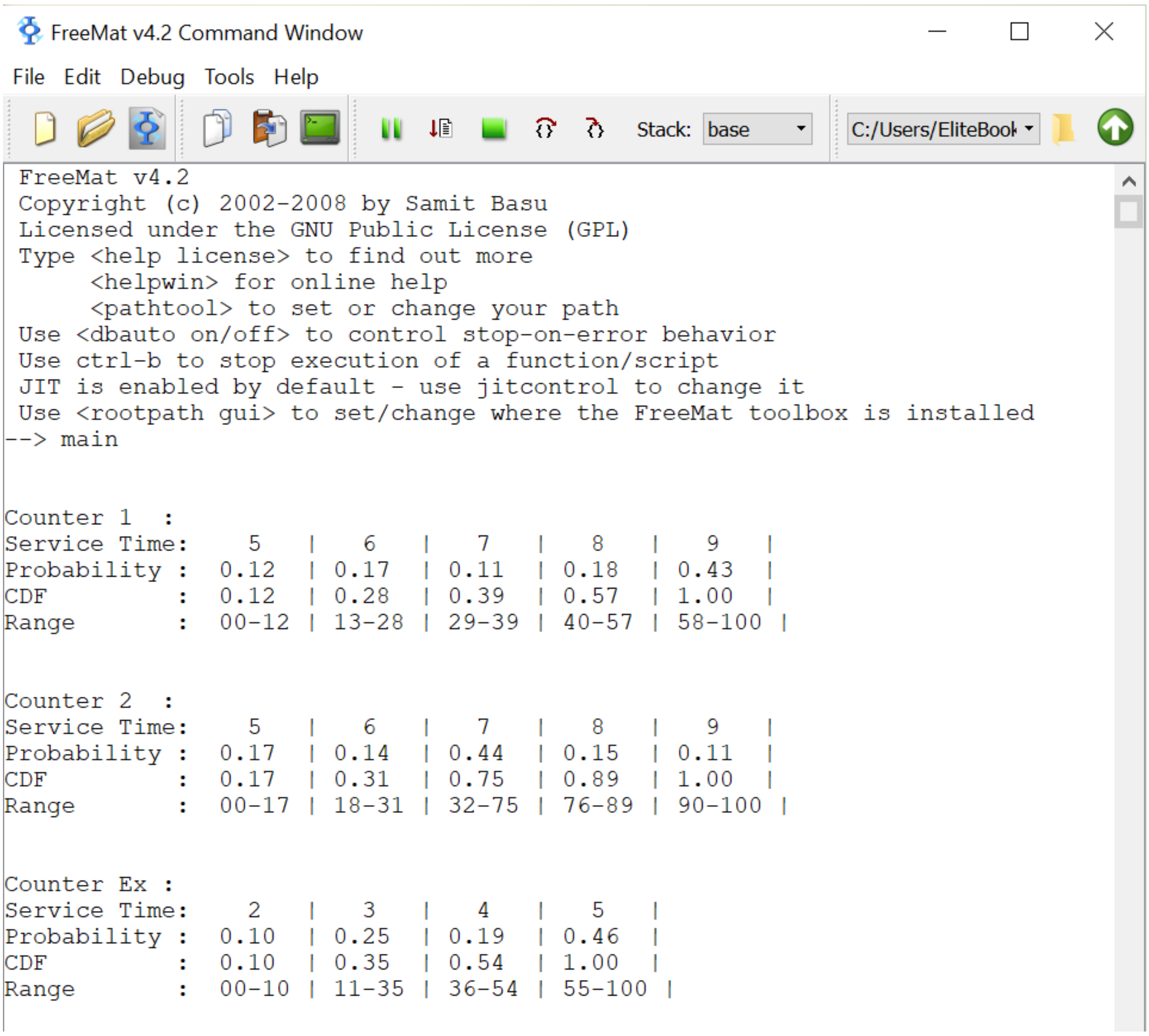
No.	Function	Purpose
1.	main.m	Main script to simulate counters and customers.
2.	mainCounter.m	Simulates counter behaviour and service time calculation.
3.	randArrayMaker.m	Generates a random array of numbers.
4.	mainCustomer.m	Simulates customer behaviour and arrival times.
5.	mainGenRand.m	Generates random numbers using different algorithms.
6.	myPause.m	Pauses execution until user input.
7.	inputCheck.m	Validates numeric user input.
8.	genLCG.m	Generates random numbers using the LCG algorithm.
9.	genRVGUD.m	Generates random numbers for uniform distribution.
10.	valueAssigner.m	Assigns values to ranges based on given data.
11.	mainWork.m	Manages simulation flow and work at counters.
12.	work.m	Simulates counter work and attribute calculations.
13.	workDisplay.m	Displays customer attributes at each counter queue.
14.	analysis.m	Performs data analysis on counter queue attributes.

3. Simulation Results

3.1 Table of Service Time, Inter-Arrival Time, and Simulation Time

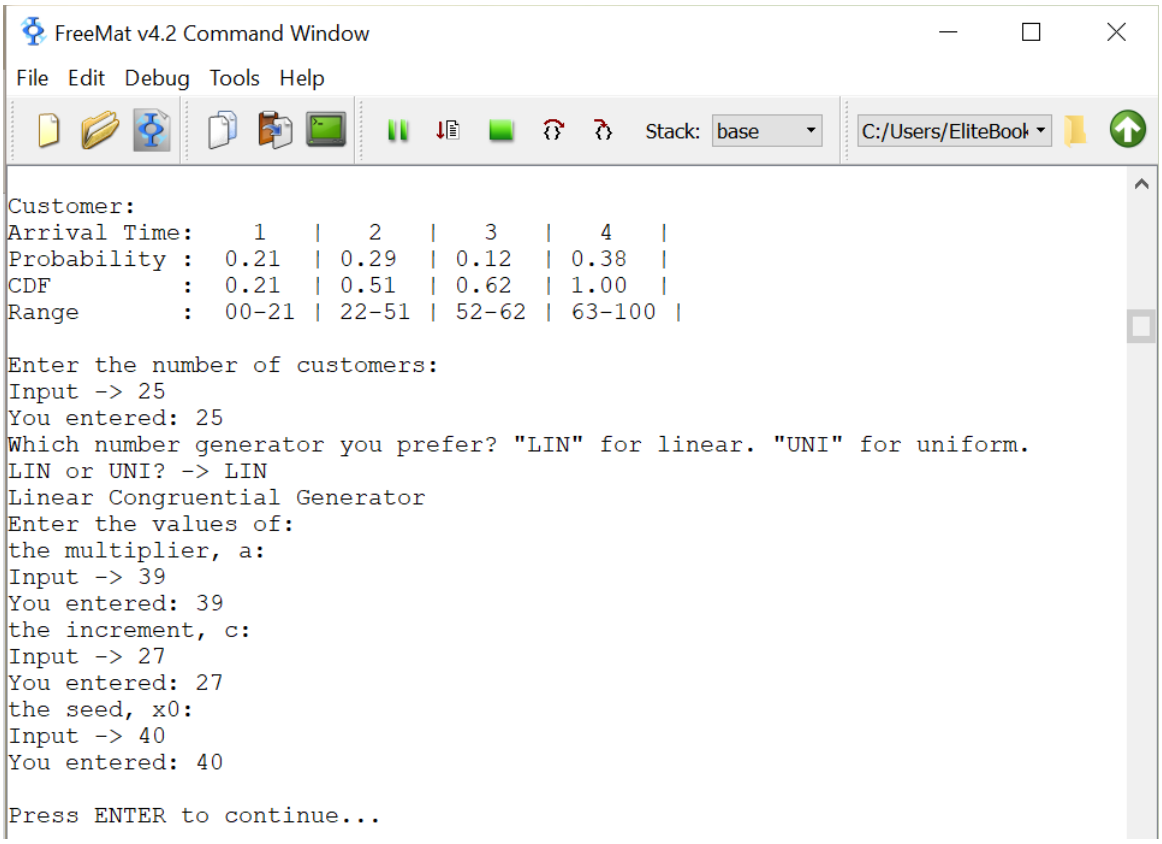
3.1.1 Linear Congruential Generator

Example Run-through on FreeMat :



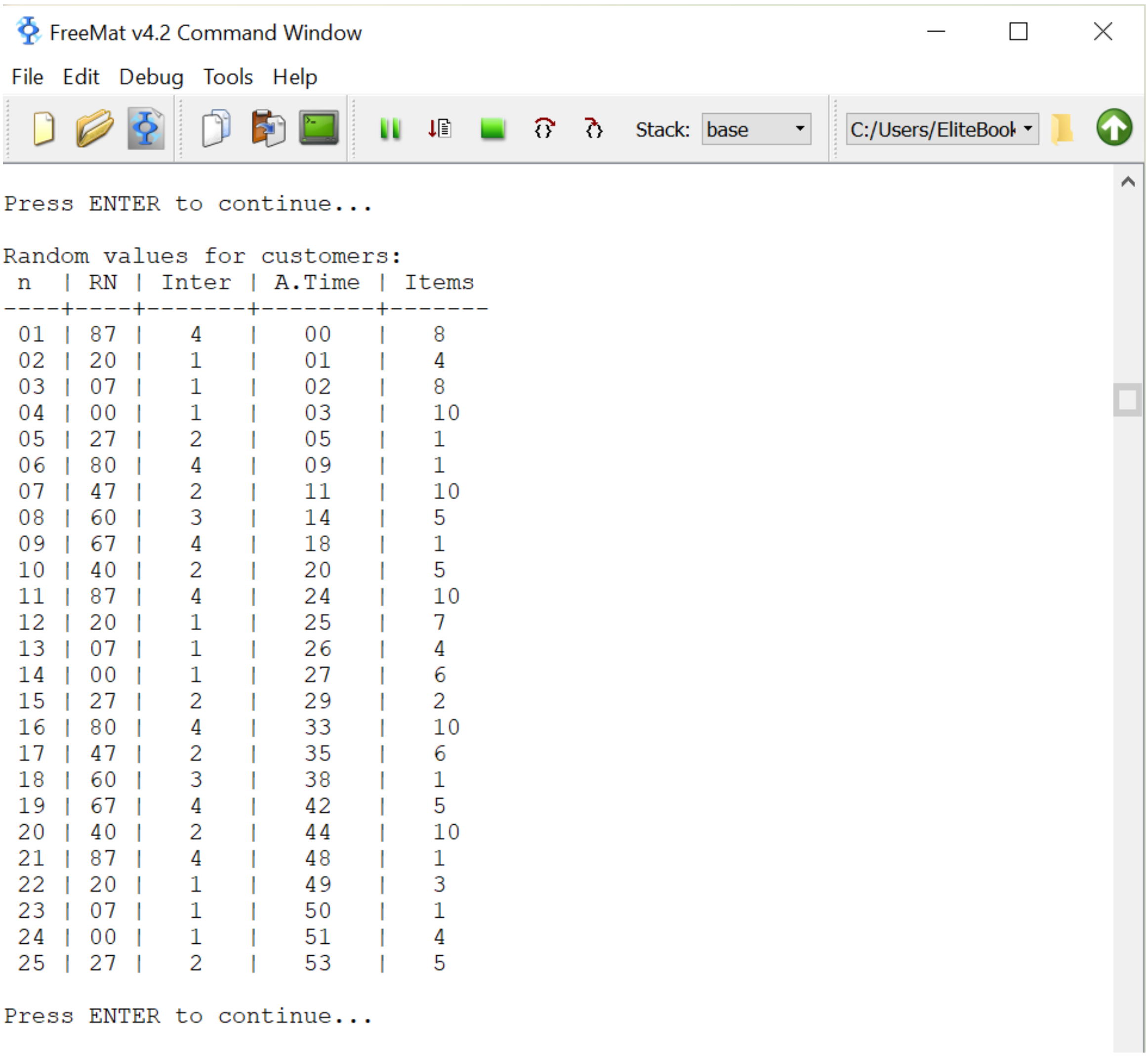
```
 FreeMat v4.2
 Copyright (c) 2002-2008 by Samit Basu
 Licensed under the GNU Public License (GPL)
 Type <help license> to find out more
      <helpwin> for online help
      <pathtool> to set or change your path
 Use <dbauto on/off> to control stop-on-error behavior
 Use ctrl-b to stop execution of a function/script
 JIT is enabled by default - use jitcontrol to change it
 Use <rootpath gui> to set/change where the FreeMat toolbox is installed
--> main


Counter 1   :
Service Time:    5    |   6    |   7    |   8    |   9    |
Probability :   0.12  |  0.17  |  0.11  |  0.18  |  0.43  |
CDF         :   0.12  |  0.28  |  0.39  |  0.57  |  1.00  |
Range       :   00-12 |  13-28 |  29-39 |  40-57 |  58-100 |


Counter 2   :
Service Time:    5    |   6    |   7    |   8    |   9    |
Probability :   0.17  |  0.14  |  0.44  |  0.15  |  0.11  |
CDF         :   0.17  |  0.31  |  0.75  |  0.89  |  1.00  |
Range       :   00-17 |  18-31 |  32-75 |  76-89 |  90-100 |


Counter Ex  :
Service Time:    2    |   3    |   4    |   5    |
Probability :   0.10  |  0.25  |  0.19  |  0.46  |
CDF         :   0.10  |  0.35  |  0.54  |  1.00  |
Range       :   00-10 |  11-35 |  36-54 |  55-100 |
```

Figure 16: Screenshot (Part 1 of 7)



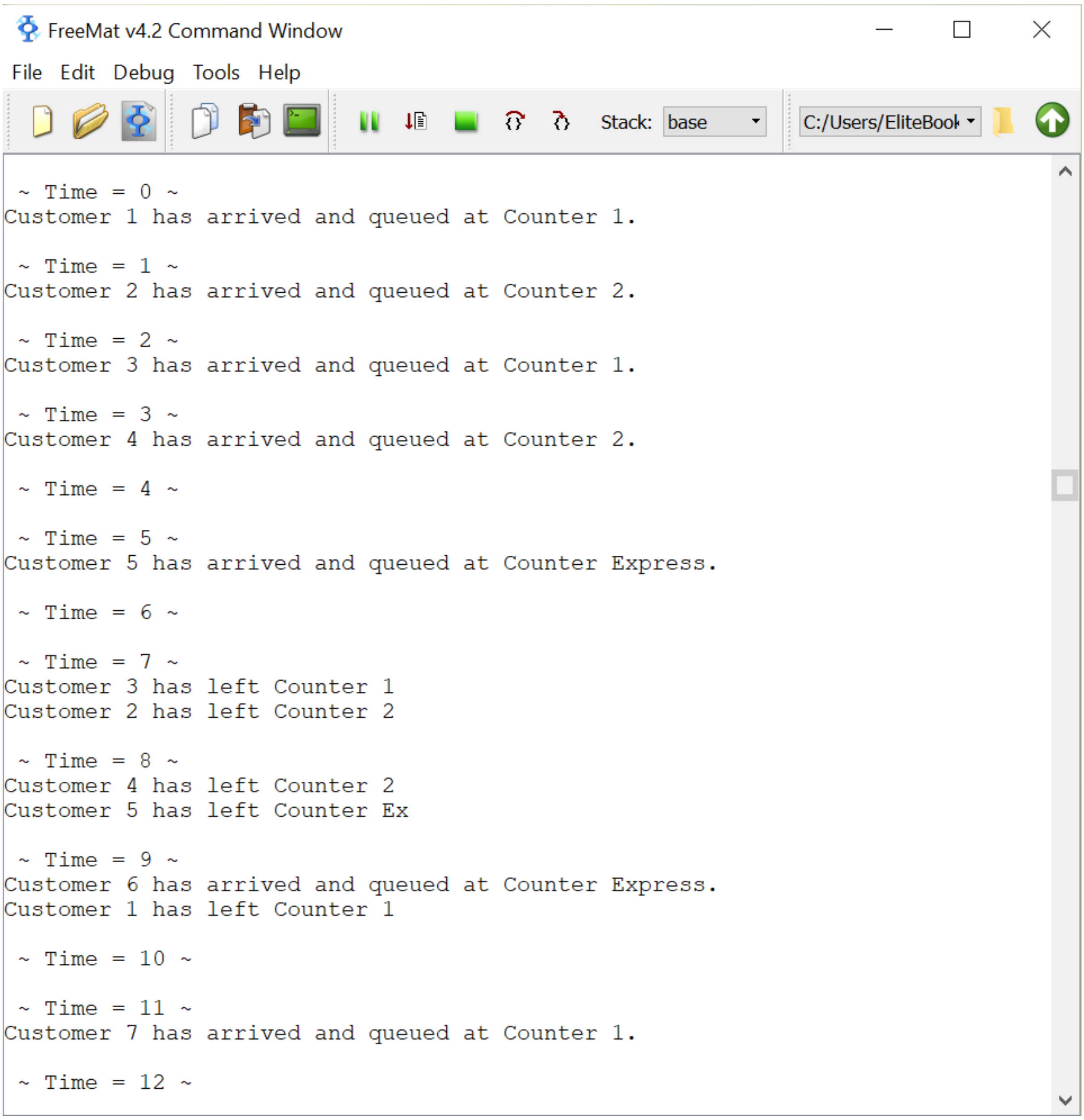
```
Customer:
Arrival Time:    1    |    2    |    3    |    4    |
Probability :  0.21   | 0.29   | 0.12   | 0.38    |
CDF         :  0.21   | 0.51   | 0.62   | 1.00    |
Range       :  00-21  | 22-51  | 52-62  | 63-100  |

Enter the number of customers:
Input -> 25
You entered: 25
Which number generator you prefer? "LIN" for linear. "UNI" for uniform.
LIN or UNI? -> LIN
Linear Congruential Generator
Enter the values of:
the multiplier, a:
Input -> 39
You entered: 39
the increment, c:
Input -> 27
You entered: 27
the seed, x0:
Input -> 40
You entered: 40

Press ENTER to continue...
```

Figure 17: Screenshot (Part 2 of 7)



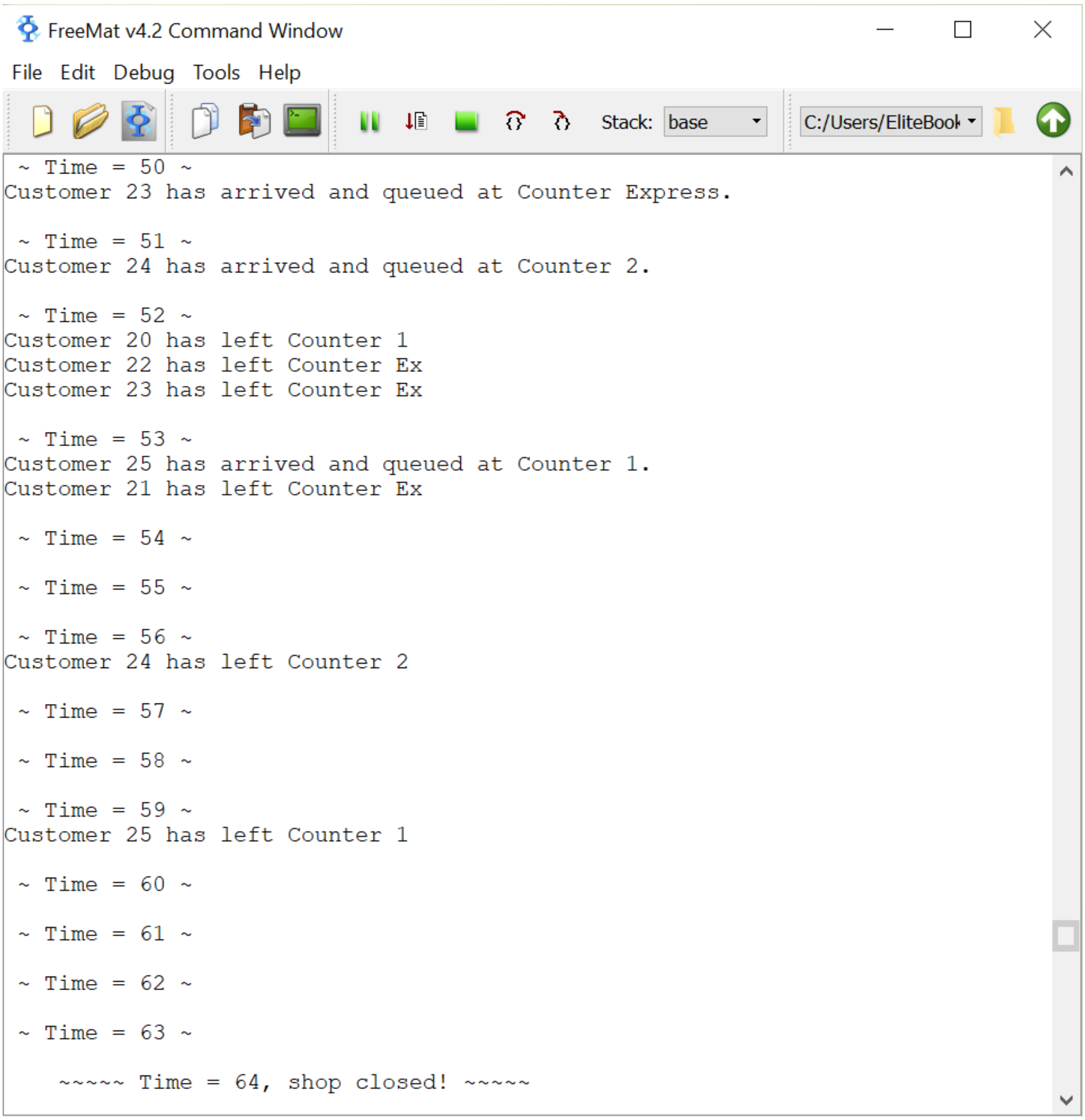
```
Press ENTER to continue...

Random values for customers:
 n  | RN | Inter | A.Time | Items
----+----+-------+--------+-------
 01 | 87 |   4   |   00   |   8
 02 | 20 |   1   |   01   |   4
 03 | 07 |   1   |   02   |   8
 04 | 00 |   1   |   03   |   10
 05 | 27 |   2   |   05   |   1
 06 | 80 |   4   |   09   |   1
 07 | 47 |   2   |   11   |   10
 08 | 60 |   3   |   14   |   5
 09 | 67 |   4   |   18   |   1
 10 | 40 |   2   |   20   |   5
 11 | 87 |   4   |   24   |   10
 12 | 20 |   1   |   25   |   7
 13 | 07 |   1   |   26   |   4
 14 | 00 |   1   |   27   |   6
 15 | 27 |   2   |   29   |   2
 16 | 80 |   4   |   33   |   10
 17 | 47 |   2   |   35   |   6
 18 | 60 |   3   |   38   |   1
 19 | 67 |   4   |   42   |   5
 20 | 40 |   2   |   44   |   10
 21 | 87 |   4   |   48   |   1
 22 | 20 |   1   |   49   |   3
 23 | 07 |   1   |   50   |   1
 24 | 00 |   1   |   51   |   4
 25 | 27 |   2   |   53   |   5

Press ENTER to continue...
```

Figure 18: Screenshot (Part 3 of 7)



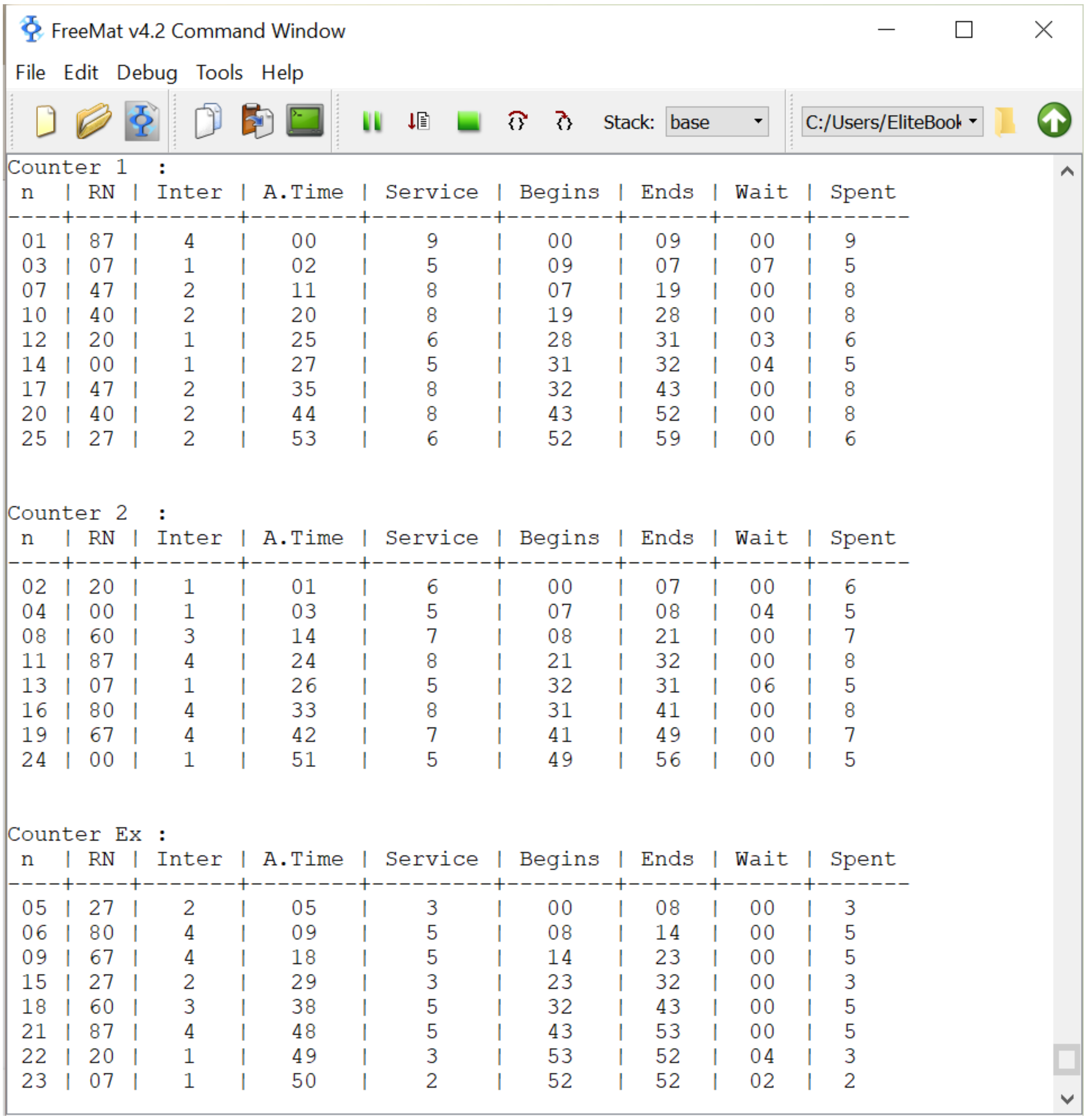
```
 ~ Time = 0 ~
Customer 1 has arrived and queued at Counter 1.

 ~ Time = 1 ~
Customer 2 has arrived and queued at Counter 2.

 ~ Time = 2 ~
Customer 3 has arrived and queued at Counter 1.

 ~ Time = 3 ~
Customer 4 has arrived and queued at Counter 2.

 ~ Time = 4 ~

 ~ Time = 5 ~
Customer 5 has arrived and queued at Counter Express.

 ~ Time = 6 ~

 ~ Time = 7 ~
Customer 3 has left Counter 1
Customer 2 has left Counter 2

 ~ Time = 8 ~
Customer 4 has left Counter 2
Customer 5 has left Counter Ex

 ~ Time = 9 ~
Customer 6 has arrived and queued at Counter Express.
Customer 1 has left Counter 1

 ~ Time = 10 ~

 ~ Time = 11 ~
Customer 7 has arrived and queued at Counter 1.

 ~ Time = 12 ~
```

Figure 19: Screenshot (Part 4 of 7)



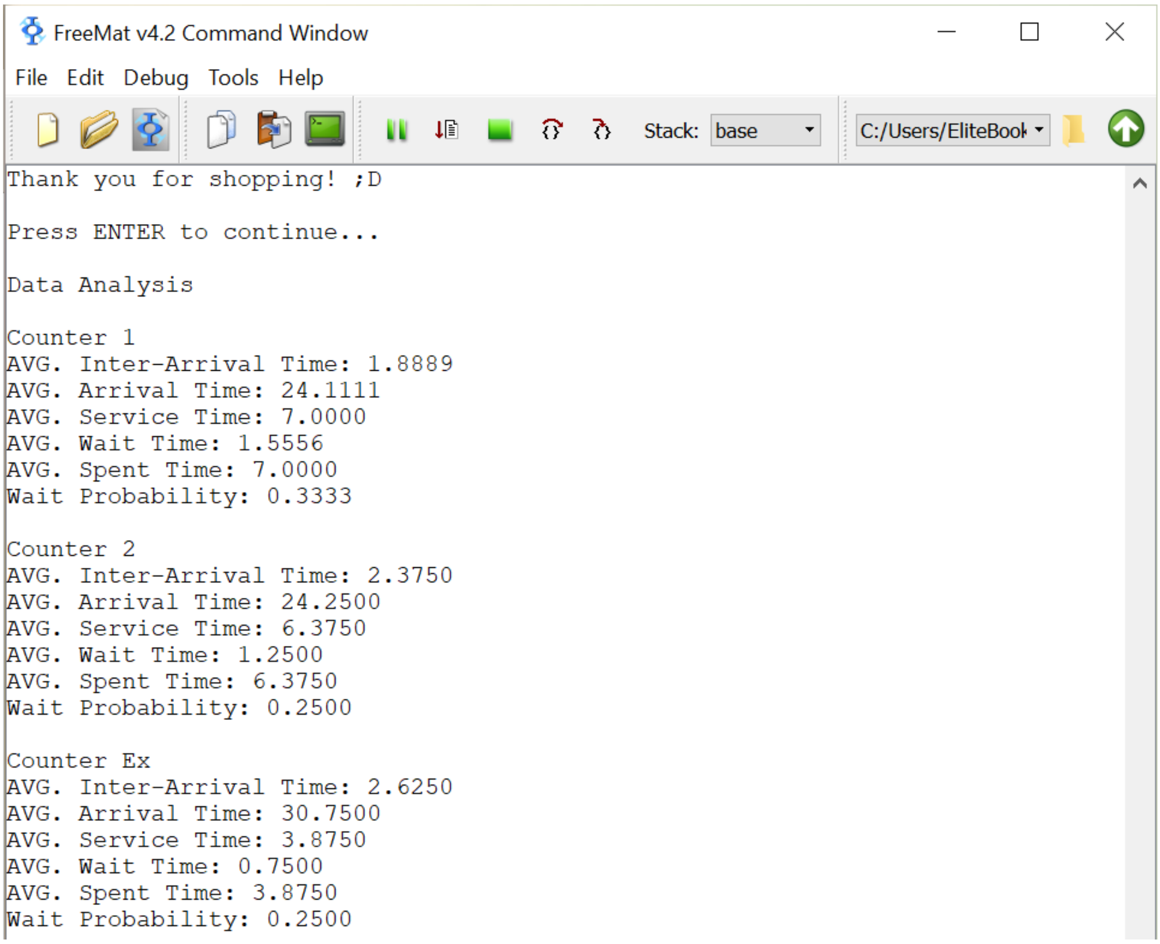
```
 ~ Time = 50 ~
Customer 23 has arrived and queued at Counter Express.

 ~ Time = 51 ~
Customer 24 has arrived and queued at Counter 2.

 ~ Time = 52 ~
Customer 20 has left Counter 1
Customer 22 has left Counter Ex
Customer 23 has left Counter Ex

 ~ Time = 53 ~
Customer 25 has arrived and queued at Counter 1.
Customer 21 has left Counter Ex

 ~ Time = 54 ~

 ~ Time = 55 ~

 ~ Time = 56 ~
Customer 24 has left Counter 2

 ~ Time = 57 ~

 ~ Time = 58 ~

 ~ Time = 59 ~
Customer 25 has left Counter 1

 ~ Time = 60 ~

 ~ Time = 61 ~

 ~ Time = 62 ~

 ~ Time = 63 ~

    ~~~~~ Time = 64, shop closed! ~~~~~
```

Figure 20: Screenshot (Part 5 of 7)



```
Counter 1  :
 n  | RN | Inter | A.Time | Service | Begins | Ends | Wait | Spent
----+----+-------+--------+---------+--------+------+------+-------
 01 | 87 |   4   |   00   |    9    |   00   |  09  |  00  |  9
 03 | 07 |   1   |   02   |    5    |   09   |  07  |  07  |  5
 07 | 47 |   2   |   11   |    8    |   07   |  19  |  00  |  8
 10 | 40 |   2   |   20   |    8    |   19   |  28  |  00  |  8
 12 | 20 |   1   |   25   |    6    |   28   |  31  |  03  |  6
 14 | 00 |   1   |   27   |    5    |   31   |  32  |  04  |  5
 17 | 47 |   2   |   35   |    8    |   32   |  43  |  00  |  8
 20 | 40 |   2   |   44   |    8    |   43   |  52  |  00  |  8
 25 | 27 |   2   |   53   |    6    |   52   |  59  |  00  |  6


Counter 2  :
 n  | RN | Inter | A.Time | Service | Begins | Ends | Wait | Spent
----+----+-------+--------+---------+--------+------+------+-------
 02 | 20 |   1   |   01   |    6    |   00   |  07  |  00  |  6
 04 | 00 |   1   |   03   |    5    |   07   |  08  |  04  |  5
 08 | 60 |   3   |   14   |    7    |   08   |  21  |  00  |  7
 11 | 87 |   4   |   24   |    8    |   21   |  32  |  00  |  8
 13 | 07 |   1   |   26   |    5    |   32   |  31  |  06  |  5
 16 | 80 |   4   |   33   |    8    |   31   |  41  |  00  |  8
 19 | 67 |   4   |   42   |    7    |   41   |  49  |  00  |  7
 24 | 00 |   1   |   51   |    5    |   49   |  56  |  00  |  5


Counter Ex :
 n  | RN | Inter | A.Time | Service | Begins | Ends | Wait | Spent
----+----+-------+--------+---------+--------+------+------+-------
 05 | 27 |   2   |   05   |    3    |   00   |  08  |  00  |  3
 06 | 80 |   4   |   09   |    5    |   08   |  14  |  00  |  5
 09 | 67 |   4   |   18   |    5    |   14   |  23  |  00  |  5
 15 | 27 |   2   |   29   |    3    |   23   |  32  |  00  |  3
 18 | 60 |   3   |   38   |    5    |   32   |  43  |  00  |  5
 21 | 87 |   4   |   48   |    5    |   43   |  53  |  00  |  5
 22 | 20 |   1   |   49   |    3    |   53   |  52  |  04  |  3
 23 | 07 |   1   |   50   |    2    |   52   |  52  |  02  |  2
```

Figure 21: Screenshot (Part 6 of 7)

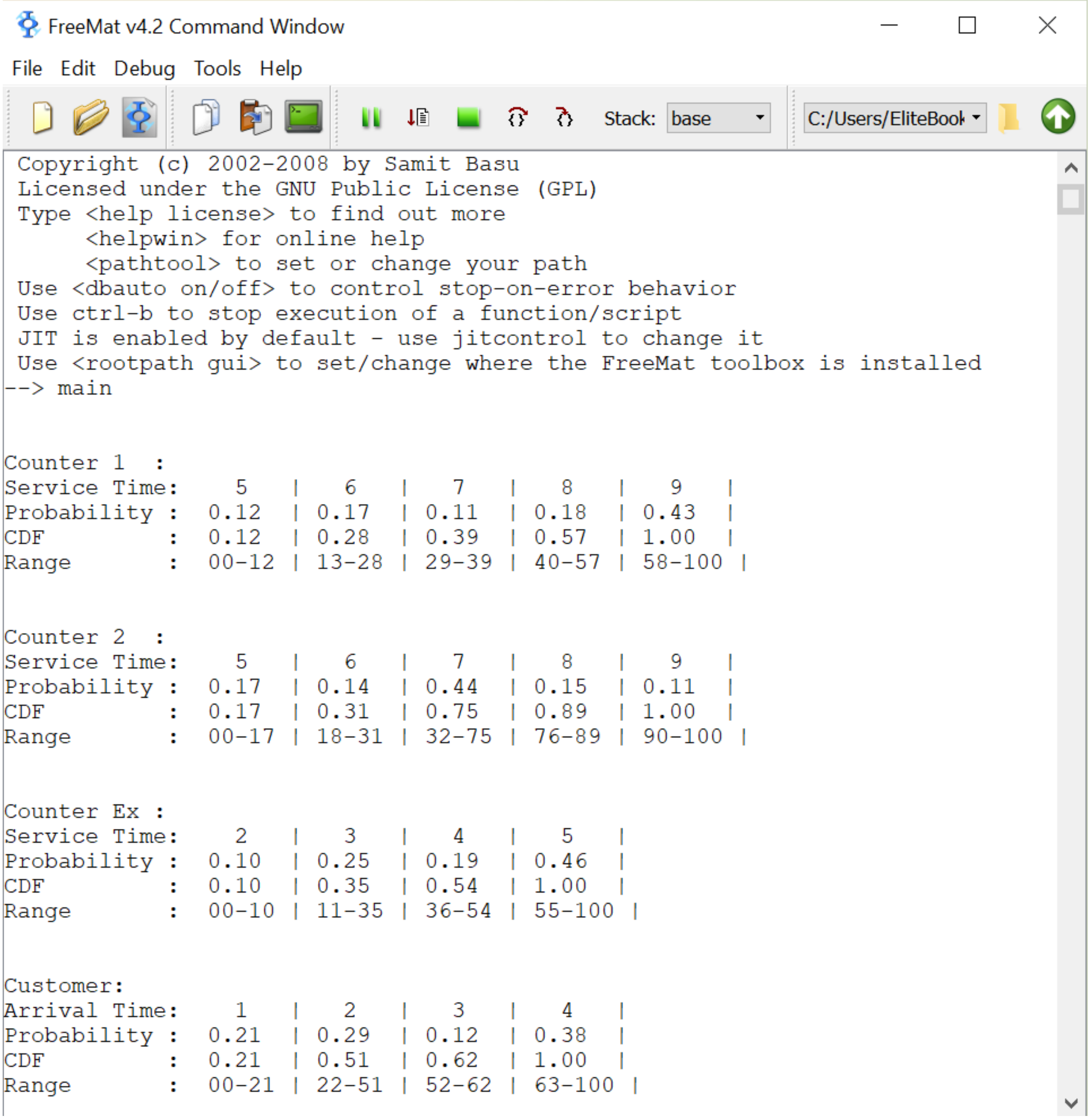


```
Thank you for shopping! ;D

Press ENTER to continue...

Data Analysis

Counter 1
AVG. Inter-Arrival Time: 1.8889
AVG. Arrival Time: 24.1111
AVG. Service Time: 7.0000
AVG. Wait Time: 1.5556
AVG. Spent Time: 7.0000
Wait Probability: 0.3333

Counter 2
AVG. Inter-Arrival Time: 2.3750
AVG. Arrival Time: 24.2500
AVG. Service Time: 6.3750
AVG. Wait Time: 1.2500
AVG. Spent Time: 6.3750
Wait Probability: 0.2500

Counter Ex
AVG. Inter-Arrival Time: 2.6250
AVG. Arrival Time: 30.7500
AVG. Service Time: 3.8750
AVG. Wait Time: 0.7500
AVG. Spent Time: 3.8750
Wait Probability: 0.2500
```

Figure 22: Screenshot (Part 7 of 7)

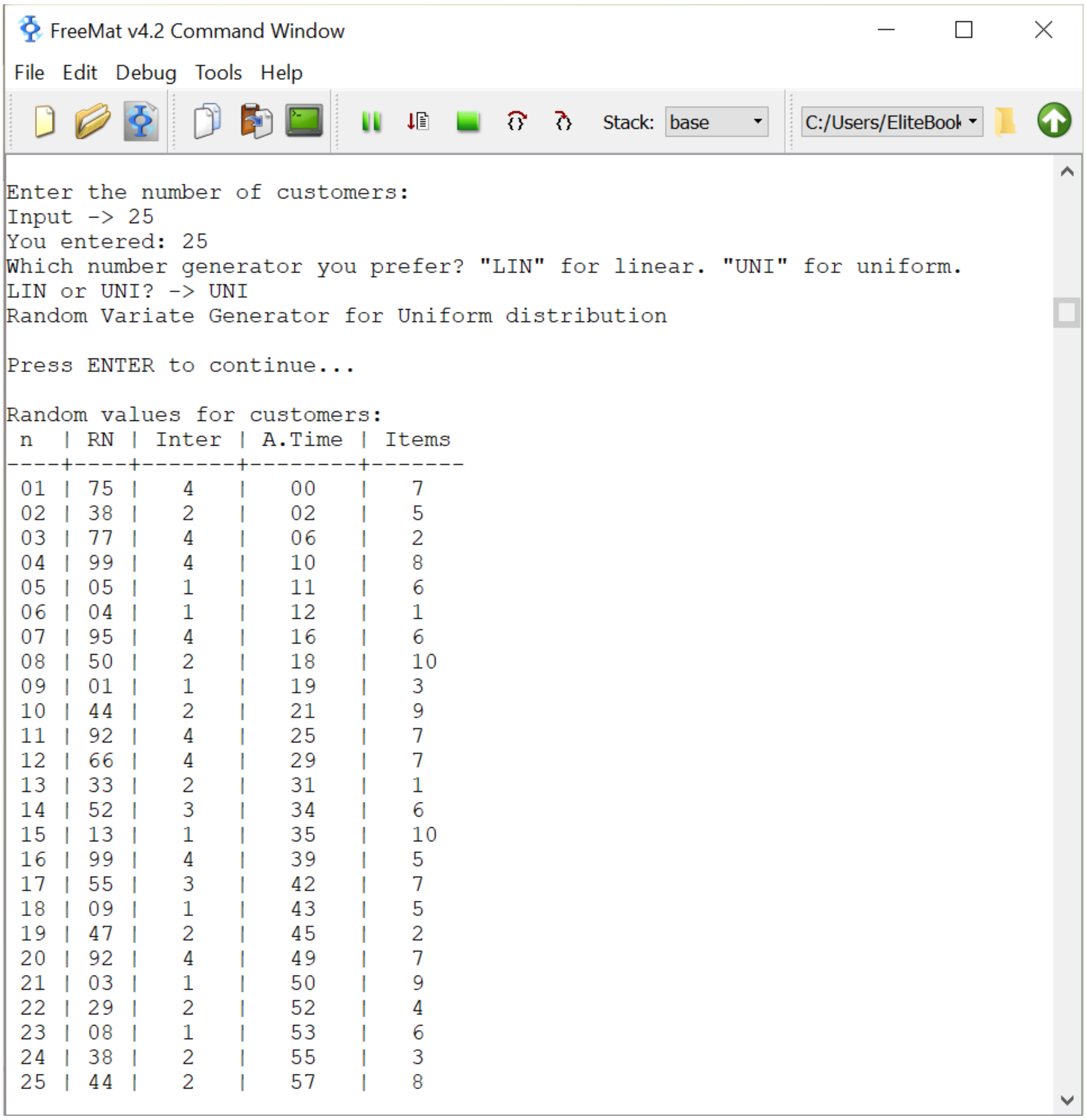
3.1.2 Random Variate Generator for Uniform Distribution

Example Run-through on FreeMat :



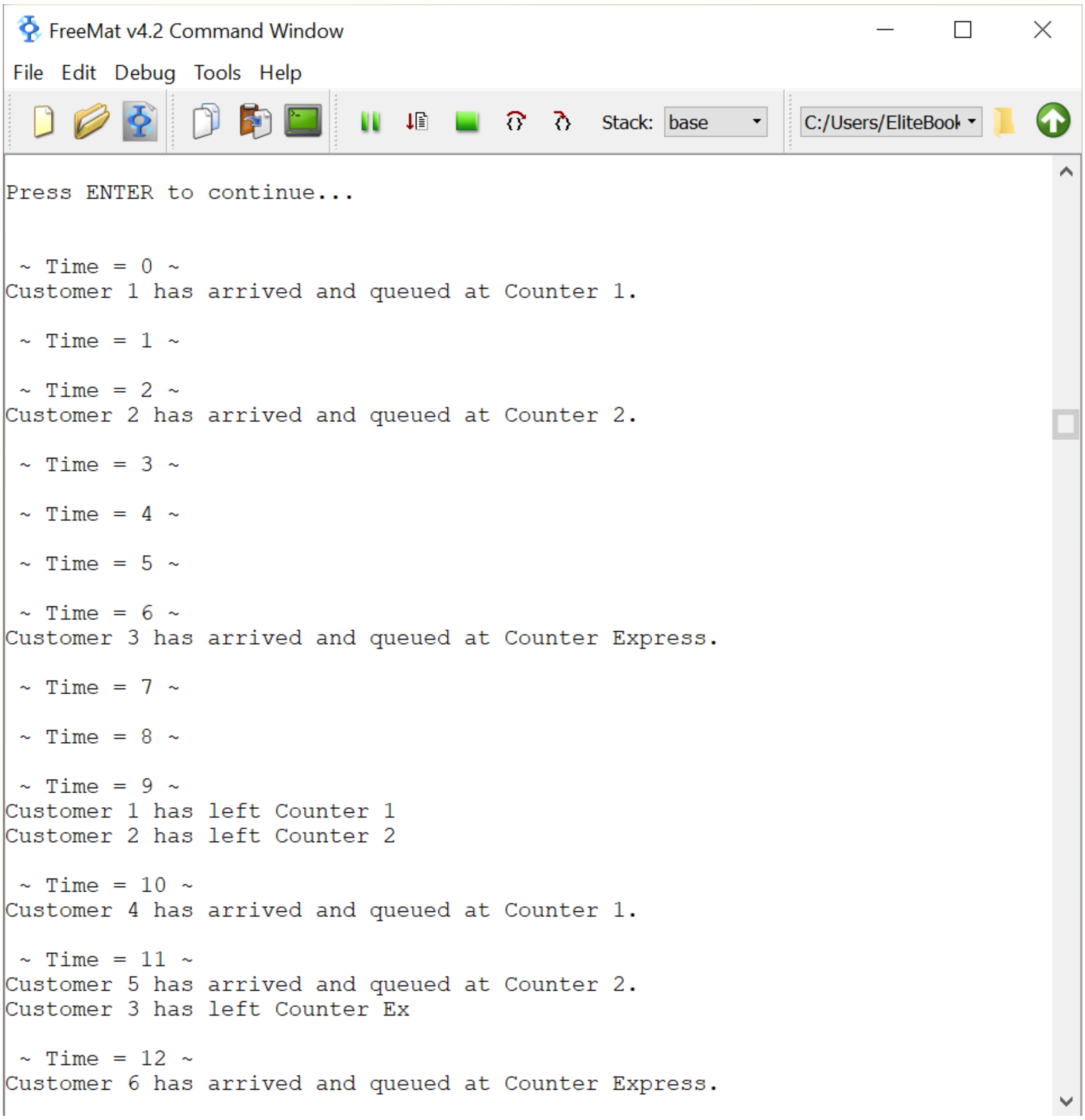
```
 Copyright (c) 2002-2008 by Samit Basu
 Licensed under the GNU Public License (GPL)
 Type <help license> to find out more
      <helpwin> for online help
      <pathtool> to set or change your path
 Use <dbauto on/off> to control stop-on-error behavior
 Use ctrl-b to stop execution of a function/script
 JIT is enabled by default - use jitcontrol to change it
 Use <rootpath gui> to set/change where the FreeMat toolbox is installed
--> main


Counter 1  :
Service Time:    5   |   6   |   7   |   8   |   9   |
Probability :  0.12  | 0.17  | 0.11  | 0.18  | 0.43  |
CDF         :  0.12  | 0.28  | 0.39  | 0.57  | 1.00  |
Range       :  00-12 | 13-28 | 29-39 | 40-57 | 58-100 |


Counter 2  :
Service Time:    5   |   6   |   7   |   8   |   9   |
Probability :  0.17  | 0.14  | 0.44  | 0.15  | 0.11  |
CDF         :  0.17  | 0.31  | 0.75  | 0.89  | 1.00  |
Range       :  00-17 | 18-31 | 32-75 | 76-89 | 90-100 |


Counter Ex :
Service Time:    2   |   3   |   4   |   5   |
Probability :  0.10  | 0.25  | 0.19  | 0.46  |
CDF         :  0.10  | 0.35  | 0.54  | 1.00  |
Range       :  00-10 | 11-35 | 36-54 | 55-100 |


Customer:
Arrival Time:    1   |   2   |   3   |   4   |
Probability :  0.21  | 0.29  | 0.12  | 0.38  |
CDF         :  0.21  | 0.51  | 0.62  | 1.00  |
Range       :  00-21 | 22-51 | 52-62 | 63-100 |
```

Figure 23: Screenshot (Part 1 of 5)



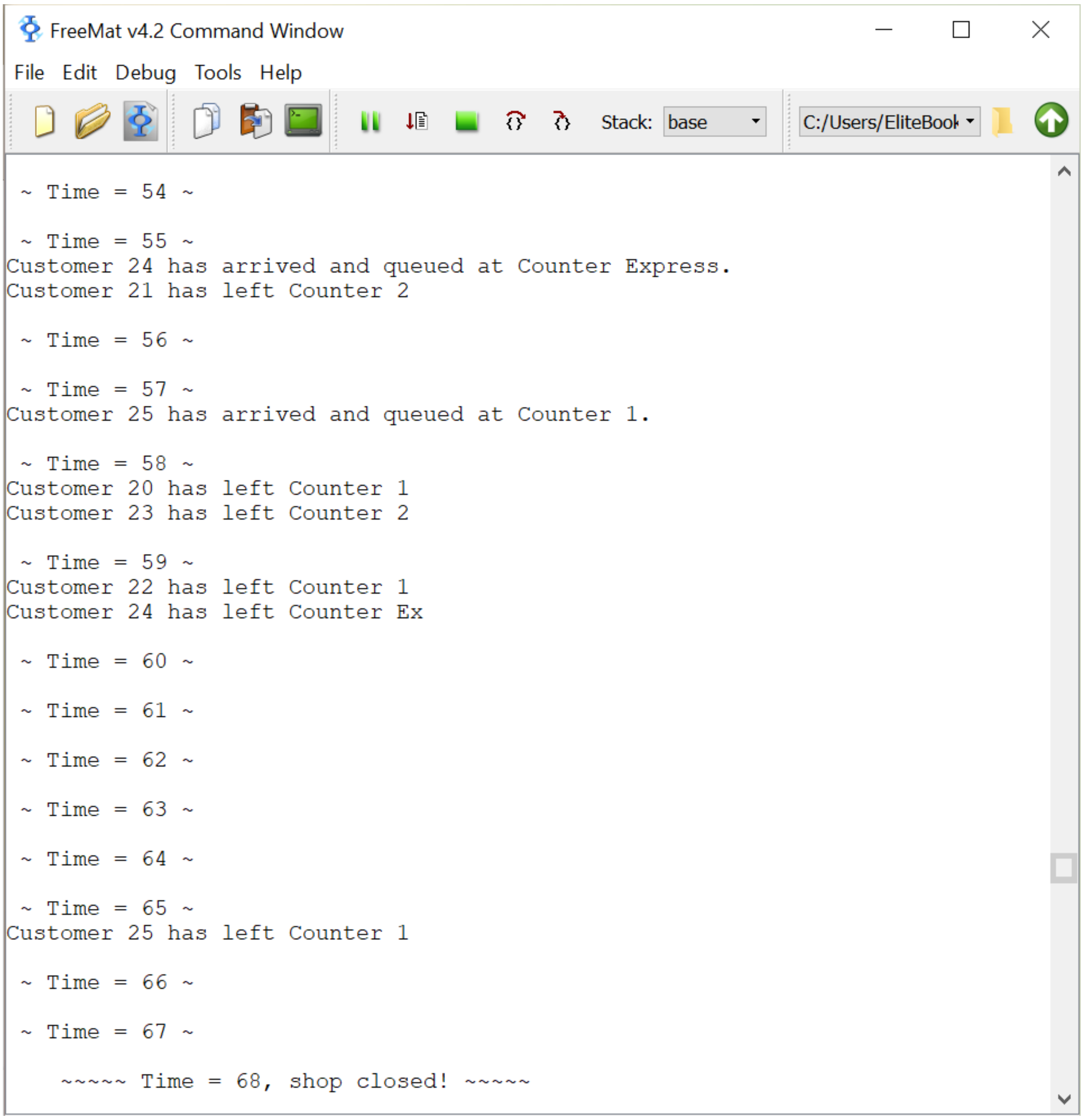
```
Enter the number of customers:
Input -> 25
You entered: 25
Which number generator you prefer? "LIN" for linear. "UNI" for uniform.
LIN or UNI? -> UNI
Random Variate Generator for Uniform distribution

Press ENTER to continue...

Random values for customers:
 n  | RN | Inter | A.Time | Items
----+----+-------+--------+-------
 01 | 75 |   4   |   00   |   7
 02 | 38 |   2   |   02   |   5
 03 | 77 |   4   |   06   |   2
 04 | 99 |   4   |   10   |   8
 05 | 05 |   1   |   11   |   6
 06 | 04 |   1   |   12   |   1
 07 | 95 |   4   |   16   |   6
 08 | 50 |   2   |   18   |   10
 09 | 01 |   1   |   19   |   3
 10 | 44 |   2   |   21   |   9
 11 | 92 |   4   |   25   |   7
 12 | 66 |   4   |   29   |   7
 13 | 33 |   2   |   31   |   1
 14 | 52 |   3   |   34   |   6
 15 | 13 |   1   |   35   |   10
 16 | 99 |   4   |   39   |   5
 17 | 55 |   3   |   42   |   7
 18 | 09 |   1   |   43   |   5
 19 | 47 |   2   |   45   |   2
 20 | 92 |   4   |   49   |   7
 21 | 03 |   1   |   50   |   9
 22 | 29 |   2   |   52   |   4
 23 | 08 |   1   |   53   |   6
 24 | 38 |   2   |   55   |   3
 25 | 44 |   2   |   57   |   8
```

Figure 24: Screenshot (Part 2 of 5)



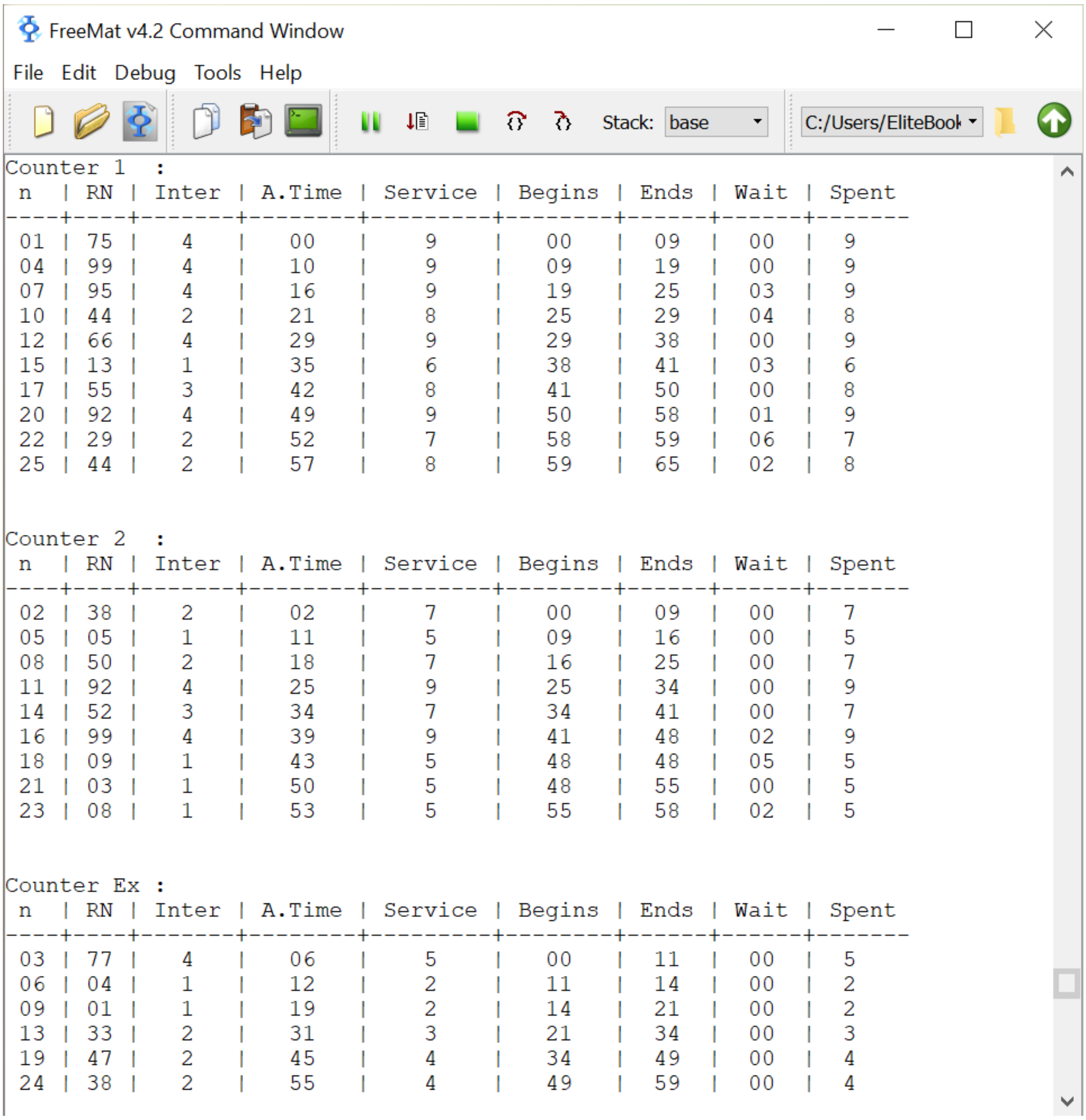
```
Press ENTER to continue...


 ~ Time = 0 ~
Customer 1 has arrived and queued at Counter 1.

 ~ Time = 1 ~

 ~ Time = 2 ~
Customer 2 has arrived and queued at Counter 2.

 ~ Time = 3 ~

 ~ Time = 4 ~

 ~ Time = 5 ~

 ~ Time = 6 ~
Customer 3 has arrived and queued at Counter Express.

 ~ Time = 7 ~

 ~ Time = 8 ~

 ~ Time = 9 ~
Customer 1 has left Counter 1
Customer 2 has left Counter 2

 ~ Time = 10 ~
Customer 4 has arrived and queued at Counter 1.

 ~ Time = 11 ~
Customer 5 has arrived and queued at Counter 2.
Customer 3 has left Counter Ex

 ~ Time = 12 ~
Customer 6 has arrived and queued at Counter Express.
```

Figure 25: Screenshot (Part 3 of 5)



```
 ~ Time = 54 ~

 ~ Time = 55 ~
Customer 24 has arrived and queued at Counter Express.
Customer 21 has left Counter 2

 ~ Time = 56 ~

 ~ Time = 57 ~
Customer 25 has arrived and queued at Counter 1.

 ~ Time = 58 ~
Customer 20 has left Counter 1
Customer 23 has left Counter 2

 ~ Time = 59 ~
Customer 22 has left Counter 1
Customer 24 has left Counter Ex

 ~ Time = 60 ~

 ~ Time = 61 ~

 ~ Time = 62 ~

 ~ Time = 63 ~

 ~ Time = 64 ~

 ~ Time = 65 ~
Customer 25 has left Counter 1

 ~ Time = 66 ~

 ~ Time = 67 ~

    ~~~~~ Time = 68, shop closed! ~~~~~
```

Figure 26: Screenshot (Part 4 of 5)



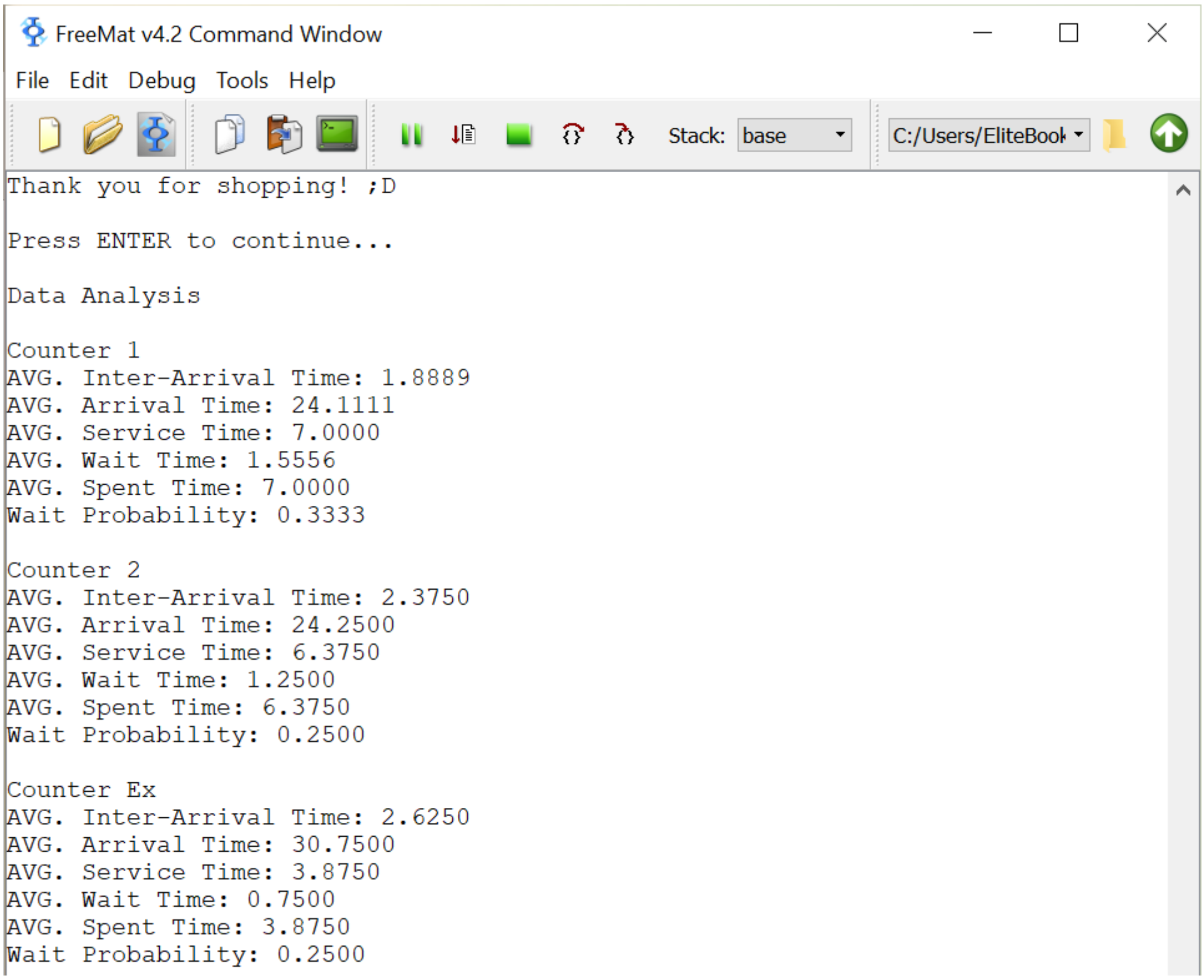
```
Counter 1  :
 n  | RN | Inter | A.Time | Service | Begins | Ends | Wait | Spent
----+----+-------+--------+---------+--------+------+------+-------
 01 | 75 |   4   |   00   |    9    |   00   |  09  |  00  |   9
 04 | 99 |   4   |   10   |    9    |   09   |  19  |  00  |   9
 07 | 95 |   4   |   16   |    9    |   19   |  25  |  03  |   9
 10 | 44 |   2   |   21   |    8    |   25   |  29  |  04  |   8
 12 | 66 |   4   |   29   |    9    |   29   |  38  |  00  |   9
 15 | 13 |   1   |   35   |    6    |   38   |  41  |  03  |   6
 17 | 55 |   3   |   42   |    8    |   41   |  50  |  00  |   8
 20 | 92 |   4   |   49   |    9    |   50   |  58  |  01  |   9
 22 | 29 |   2   |   52   |    7    |   58   |  59  |  06  |   7
 25 | 44 |   2   |   57   |    8    |   59   |  65  |  02  |   8


Counter 2  :
 n  | RN | Inter | A.Time | Service | Begins | Ends | Wait | Spent
----+----+-------+--------+---------+--------+------+------+-------
 02 | 38 |   2   |   02   |    7    |   00   |  09  |  00  |   7
 05 | 05 |   1   |   11   |    5    |   09   |  16  |  00  |   5
 08 | 50 |   2   |   18   |    7    |   16   |  25  |  00  |   7
 11 | 92 |   4   |   25   |    9    |   25   |  34  |  00  |   9
 14 | 52 |   3   |   34   |    7    |   34   |  41  |  00  |   7
 16 | 99 |   4   |   39   |    9    |   41   |  48  |  02  |   9
 18 | 09 |   1   |   43   |    5    |   48   |  48  |  05  |   5
 21 | 03 |   1   |   50   |    5    |   48   |  55  |  00  |   5
 23 | 08 |   1   |   53   |    5    |   55   |  58  |  02  |   5


Counter Ex :
 n  | RN | Inter | A.Time | Service | Begins | Ends | Wait | Spent
----+----+-------+--------+---------+--------+------+------+-------
 03 | 77 |   4   |   06   |    5    |   00   |  11  |  00  |   5
 06 | 04 |   1   |   12   |    2    |   11   |  14  |  00  |   2
 09 | 01 |   1   |   19   |    2    |   14   |  21  |  00  |   2
 13 | 33 |   2   |   31   |    3    |   21   |  34  |  00  |   3
 19 | 47 |   2   |   45   |    4    |   34   |  49  |  00  |   4
 24 | 38 |   2   |   55   |    4    |   49   |  59  |  00  |   4
```

Figure 27: Screenshot (Part 5 of 5)

3.2 Evaluation Results

- Average Inter-Arrival Time
- Average Arrival Time
- Average Service Time
- Average Waiting Time
- Average Spent Time
- Wait Probability

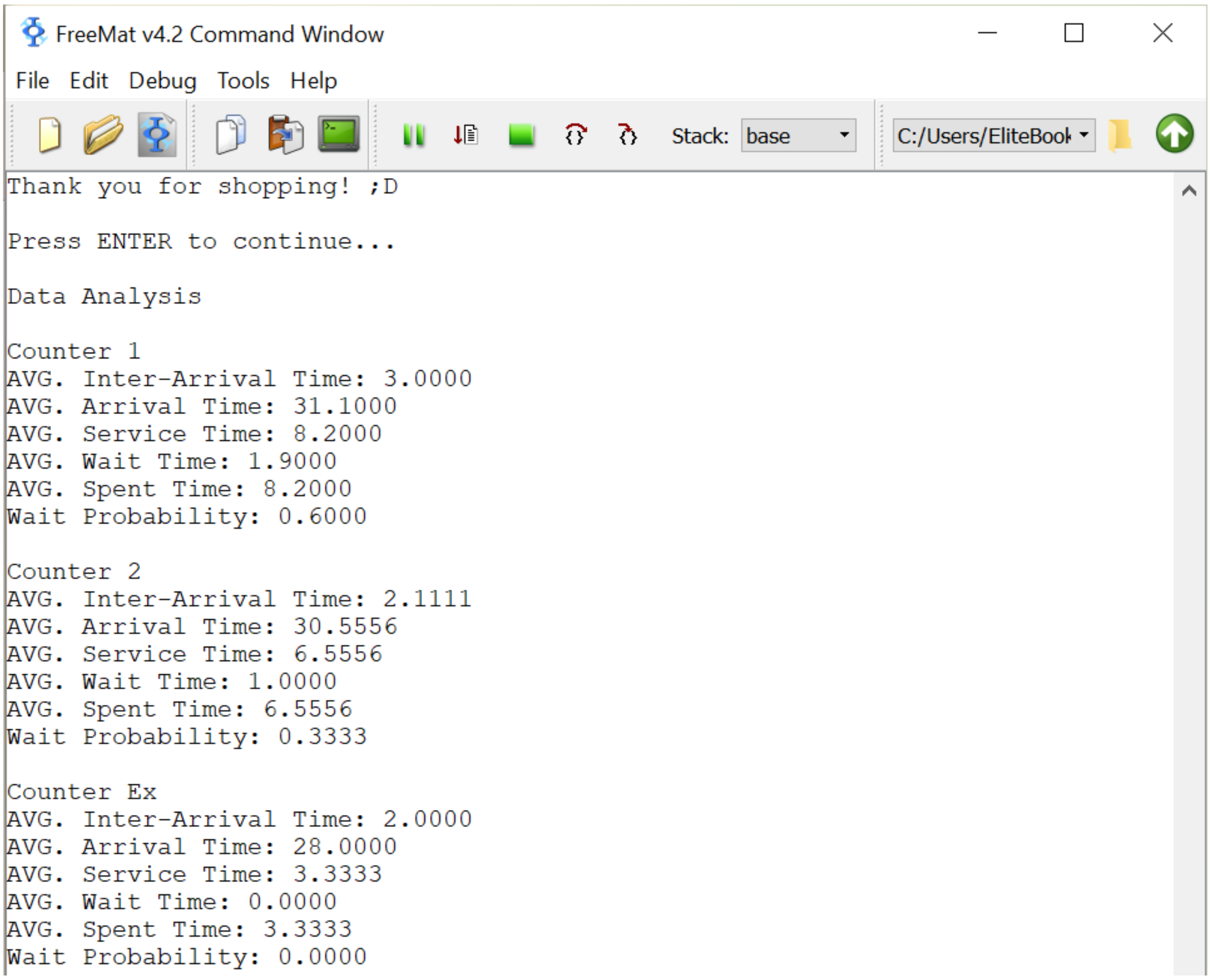
3.2.1 Data Analysis of **Linear Congruential Generator:**



```
Thank you for shopping! ;D

Press ENTER to continue...

Data Analysis

Counter 1
AVG. Inter-Arrival Time: 1.8889
AVG. Arrival Time: 24.1111
AVG. Service Time: 7.0000
AVG. Wait Time: 1.5556
AVG. Spent Time: 7.0000
Wait Probability: 0.3333

Counter 2
AVG. Inter-Arrival Time: 2.3750
AVG. Arrival Time: 24.2500
AVG. Service Time: 6.3750
AVG. Wait Time: 1.2500
AVG. Spent Time: 6.3750
Wait Probability: 0.2500

Counter Ex
AVG. Inter-Arrival Time: 2.6250
AVG. Arrival Time: 30.7500
AVG. Service Time: 3.8750
AVG. Wait Time: 0.7500
AVG. Spent Time: 3.8750
Wait Probability: 0.2500
```

Figure 28: Screenshot of Data Analysis for LCG

3.2.2 Data Analysis of **Random Variate Generator for Uniform Distribution:**



```
Thank you for shopping! ;D

Press ENTER to continue...

Data Analysis

Counter 1
AVG. Inter-Arrival Time: 3.0000
AVG. Arrival Time: 31.1000
AVG. Service Time: 8.2000
AVG. Wait Time: 1.9000
AVG. Spent Time: 8.2000
Wait Probability: 0.6000

Counter 2
AVG. Inter-Arrival Time: 2.1111
AVG. Arrival Time: 30.5556
AVG. Service Time: 6.5556
AVG. Wait Time: 1.0000
AVG. Spent Time: 6.5556
Wait Probability: 0.3333

Counter Ex
AVG. Inter-Arrival Time: 2.0000
AVG. Arrival Time: 28.0000
AVG. Service Time: 3.3333
AVG. Wait Time: 0.0000
AVG. Spent Time: 3.3333
Wait Probability: 0.0000
```

Figure 29: Screenshot of Data Analysis for RVG for Uniform Distribution

4. Conclusion

In conclusion, the development of a queueing simulator using FreeMat has enhanced our understanding of queuing systems and provided practical insights into their performance. By combining theoretical knowledge with hands-on implementation, we have acquired valuable skills in coding, simulation, and analysis of data. We strongly believe that this assignment has laid a solid foundation for future applications of queuing theory and Monte Carlo simulations in problem-solving and decision-making processes.

5. References

1) Freemat Documentation : https://freemat.sourceforge.net/help/index.html
2) Flowchart Maker: https://www.drawio.com/