Lab Report

TERM:	**2023-2024**
Module:	**EE2634 Digital Systems Design and Reliability Engineering**
CLASS:	34092102
BRUNEL ID:	2161047
NAME:	Xukang Liu
TUTOR:	Liangbo Xie

30, Nov 2023

Lab 1: Design and Simulation of Digital Comparators

1. Introduction

This report comprehensively outlines the design methodology and simulation procedures employed for both the 2-bit and 4-bit digital comparators. The primary objective of these experiments is to showcase and scrutinize the comprehension of fundamental concepts and practical applications within digital circuits and systems. Moreover, the experiments aim to evaluate the proficiency in designing and simulating digital circuits by utilizing learned methods and Computer-Aided Design (CAD) tools.

2. Experiment 1: Design and Simulation of a 2-bit Binary Digital Comparator

2.1. Experimental design ideas

This experiment centers on designing a 2-bit comparator circuit with inputs A ($a_1 a_0$) and B ($b_1 b_0$), generating three distinct outputs: A>B (AgtB), A<B (AltB), and A=B (AeqB). The objective is to ascertain the logic expressions for the corresponding combinational logic circuit, enabling results through VHDL simulation. To achieve this efficiently within the constraints of limited bits, we'll create a Karnaugh map based on the truth table to derive the most straightforward logical expressions. Additionally, the comparison logic can be arranged in priority from the highest to the lowest according to established comparison principles.

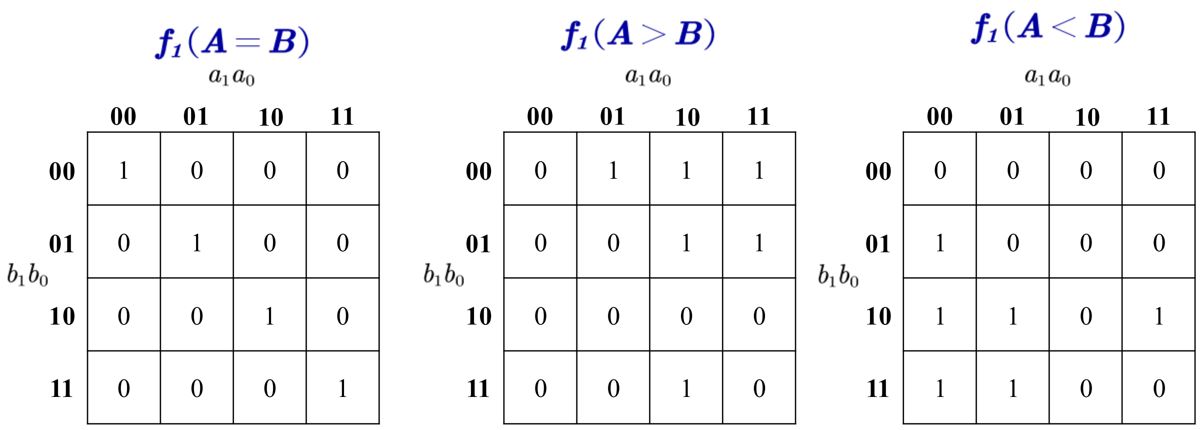
2.2. Deriving logical expressions

(1) Method 1: Logical expressions are deduced through truth tables

Table 1: Truth table of 2-bit comparator

Input				**Output**		
a_1	**a_0**	**b_1**	**b_0**	**AgtB(A>B)**	**AeqB(A=B)**	**AltB(A<B)**
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0

1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0


$f_1(A=B)$
$f_1(A>B)$
$f_1(A<B)$

$$f_1(A=B)$$

b_1b_0 \ a_1a_0	00	01	10	11
00	1	0	0	0
01	0	1	0	0
10	0	0	1	0
11	0	0	0	1

$$f_1(A>B)$$

b_1b_0 \ a_1a_0	00	01	10	11
00	0	1	1	1
01	0	0	1	1
10	0	0	0	0
11	0	0	1	0

$$f_1(A<B)$$

b_1b_0 \ a_1a_0	00	01	10	11
00	0	0	0	0
01	1	0	0	0
10	1	1	0	1
11	1	1	0	0

Figure 1: K-map of 2-bit comparator

Figure 1 shows the corresponding k-map of 2-bit comparator. From this K-map, the corresponding logical expression can then be derived.

$$\begin{aligned} f_1(A=B) &= \overline{a}_1\overline{a}_0\overline{b}_1\overline{b}_0 + \overline{a}_1 a_0 \overline{b}_1 b_0 + a_1\overline{a}_0 b_1\overline{b}_0 + a_1 a_0 b_1 b_0 \\ &= \left(a_1 b_1 + \overline{a}_1\overline{b}_0\right)\left(a_0 b_0 + \overline{a}_0\overline{b}_0\right) \\ &= \left(\overline{a_1 \oplus b_1}\right)\left(\overline{a_0 \oplus b_0}\right) \end{aligned}$$

$$\begin{aligned} f_2(A>B) &= a_1\overline{b}_1 + a_1 a_0 \overline{b}_0 + a_0\overline{b}_1\overline{b}_0 \\ &= a_1\overline{b}_1 + a_1 a_0 \overline{b}_0 + a_0\overline{b}_1\overline{b}_0 \\ &= a_1\overline{b}_1 + a_0\overline{b}_0\left(a_1 + \overline{b}_1\right) \\ &= a_1\overline{b_1} + a_0\overline{b_0}\left(\overline{a_1 \oplus b_1}\right) \end{aligned}$$

$$
\begin{aligned}
f_3(A<B) &= \overline{a}_1 b_1 + \overline{a}_1 \overline{a}_0 \overline{b}_1 b_0 + a_1 \overline{a}_0 b_1 b_0 \\
&= \overline{a}_1 b_1 + \overline{a}_0 b_0 \left(\overline{a}_1 \overline{b}_1 + a_1 b_1 \right) \\
&= \overline{a}_1 b_1 + \overline{a}_0 b_0 \left(\overline{a_1 \oplus b_1} \right)
\end{aligned}
$$

(2) Method 2: Consider the bits of A and B in pairs to derive the comparator circuit

When comparing magnitudes, prioritization occurs **from the most significant to the least significant bits**. In other words, the significance of the high bit takes precedence over the low bit in establishing priority among the bits being compared.

Define a set of intermediate signals called i_1, i_0. Each signal, i_k, is 1 if the bits of A and B with the same index are equal. That is, $i_k = \overline{a_k \oplus b_k}$. The comparator's *AeqB* output is then given by

$$AeqB = i_1 i_0$$

The *AgtB* output can be obtained by evaluating the bits of A and B in sequence, starting from the most significant bit to the least significant bit. The initial differing bit-position, denoted as *k*, between a_k and b_k determines whether A is greater than or less than B. *AgtB* can be expressed as

$$AgtB = a_1 \overline{b}_1 + i_1 a_0 \overline{b}_0$$

The *AltB* output can be derived by using the other two output as

$$AltB = \overline{AeqB + AgtB}$$

After simplifying method 1, it is found that the **logical expressions of method 1 and method 2 are the same.**

2.3. Reduces to the form XNOR, NAND, NOR

The logical expressions of these three gates are: $XNOR = \overline{a \oplus b}$, $NAND = \overline{a.b}$, $NOR = \overline{a+b}$. Using De Morgan's theorem, it can be reduced to the desired form.

$$AeqB = \left(\overline{a_1 \oplus b_1}\right)\left(\overline{a_0 \oplus b_0}\right)$$

$$= \overline{\left[\overline{\left(\overline{a_1 \oplus b_1}\right)\left(\overline{a_0 \oplus b_0}\right)}\right] . \left[\overline{\left(\overline{a_1 \oplus b_1}\right)\left(\overline{a_0 \oplus b_0}\right)}\right]}$$

$$AgtB = a_1\,\overline{b}_1 + a_0\,\overline{b}_0\left(\overline{a_1 \oplus b_1}\right)$$

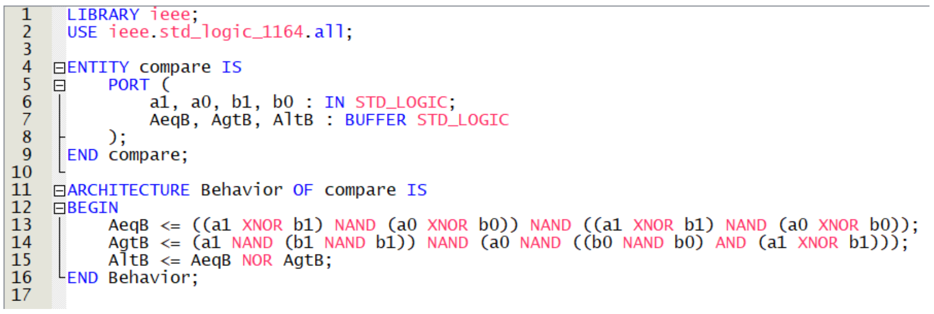
$$= \overline{\overline{a_1\,\overline{b_1} + a_0\,\overline{b_0}\left(\overline{a_1 \oplus b_1}\right)}}$$

$$= \overline{\overline{a_1\,\overline{b_1}}.\,\overline{a_0\,\overline{b_0}\left(\overline{a_1 \oplus b_1}\right)}}$$

$$AltB = \overline{AeqB + AgtB}$$

2.4. Simulation and Results analysis

Using CAD tools, write VHDL codes to simulate the logical expression. In this experiment, I use Quartus ii to simulate it.



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY compare IS
    PORT (
        a1, a0, b1, b0 : IN STD_LOGIC;
        AeqB, AgtB, AltB : BUFFER STD_LOGIC
    );
END compare;

ARCHITECTURE Behavior OF compare IS
BEGIN
    AeqB <= ((a1 XNOR b1) NAND (a0 XNOR b0)) NAND ((a1 XNOR b1) NAND (a0 XNOR b0));
    AgtB <= (a1 NAND (b1 NAND b1)) NAND (a0 NAND ((b0 NAND b0) AND (a1 XNOR b1)));
    AltB <= AeqB NOR AgtB;
END Behavior;
```

Figure 2: Code section

Figure 2 displays the code I composed for the 2-bit comparator. As observed, all three logic gates—NAND, NOR, and XNOR—are utilized in this implementation. In the logical expression, there are NOT gates, and the same function can be achieved by two NAND gates in the code.

In the segment responsible for *AgtB*, I employed an AND gate, which is a substitution for the NAND3 gate utilized in the logical expression. It's worth noting that while the VHDL expression doesn't explicitly represent a NAND3 gate, its functionality can be emulated through the use of an AND gate in the VHDL code. In practical circuitry, however, the logical operation specified by the NAND3 gate can be effectively realized.

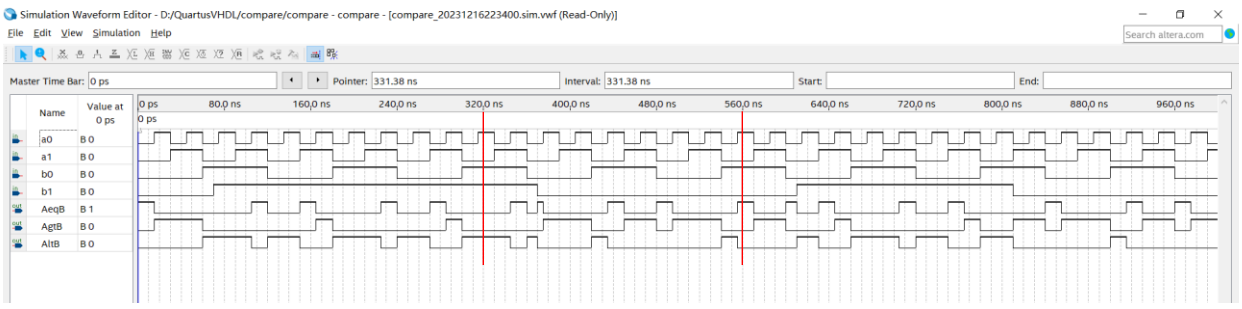


Figure 3: 2-bit comparator waveform

Figure 3 depicts the waveform plot associated with the bit comparator. Upon examining the waveform diagram, it's evident that at distinct time intervals, precisely one of the three states—

AeqB, *AgtB*, or *AltB*—remains consistently at a high level. This initial observation provides an initial indication that the simulation results are likely accurate. Furthermore, to validate the simulation results more comprehensively, I specifically focused on two-time instances: 320.0ns and 560.0ns. These moments were chosen for further scrutiny to confirm the correctness and accuracy of my simulation outcomes.

For $t = 320.0ns$, $a_1a_0 = 01$, $b_1b_0 = 11$, so $B > A$. From the waveform diagram, *AltB* is in the position of high level, so the simulation results are correct.

For $t = 560.0ns$, $a_1a_0 = 01$, $b_1b_0 = 01$, so $A = B$. From the waveform diagram, *AeqB* is in the position of high level, so the simulation results are correct.

2.5. Circuit diagram

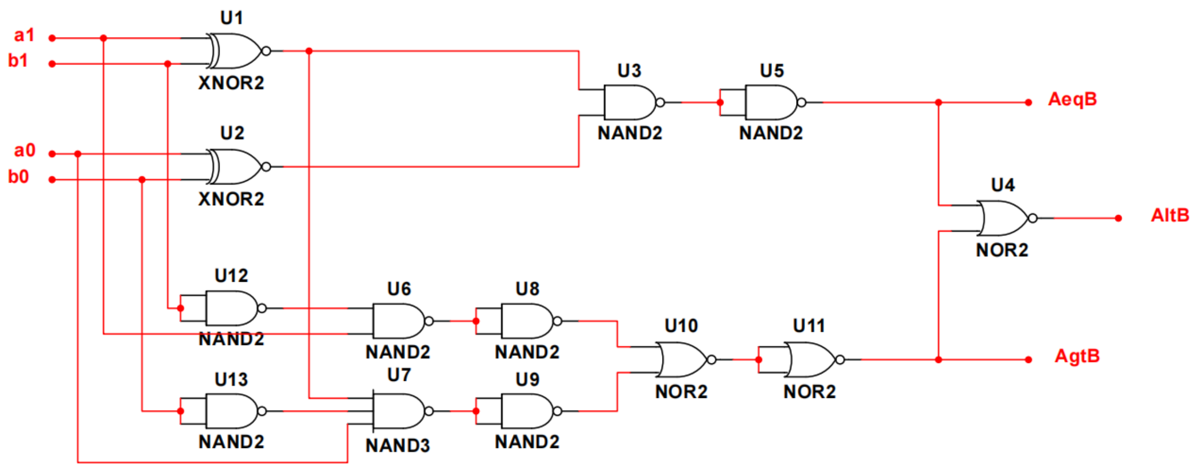


Figure 4: Circuit diagram

Figure 4 shows the corresponding 2-bit comparator circuit diagram. In the circuit diagram, I only used NAND, XNOR and NOR to achieve the function of comparison.

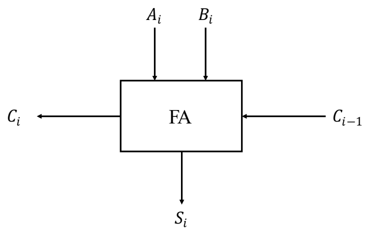
3. Experiment 2: Design and Simulation of a 4-bit Binary Digital Comparator

3.1. Experimental design ideas

The truth table for the 4-bit comparator involves 8 bits in total, resulting in a total of 256 cases. Given the sheer volume of cases, deriving the logical expressions directly from the truth table becomes exceedingly complex and impractical. Additionally, constructing a Karnaugh map for this scenario is unfeasible due to the extensive number of combinations involved.

Another way to compare two numbers is to use the magnitude of the subtraction of two numbers as the principle of comparison. Let $A = a_3a_2a_1a_0$, $B = b_3b_2b_1b_0$. *AeqB* means $A - B = 0$, *AgtB* means $A - B > 0$, *AltB* means $A - B < 0$. Hence, utilizing a subtractor enables a comparison of sizes, where the three distinct output states of the subtractor can serve as the foundation for establishing the relative magnitudes or sizes of the compared values [1]. $A - B = A + (-B)$, therefore, a 4bit comparator can be built **using a full adder** instead of a subtractor [2].

3.2. Full adder explanation



The left figure shows the structure diagram of full adder. Both of A_i *and* B_i are addend, C_{i-1} is low bit carry, C_i is the high bit carry and S_i is sum. The relationship between them can be written by logical expressions.

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = A_i B_i + C_{i-1}(A_i \oplus B_i)$$

Table 2: Truth table of full-adder

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3.3. Design of 4bit comparator circuit

When employing **cascaded multiple full adders**, it's crucial to consider the impact of carry propagation. One notable advantage of utilizing full adders lies in their ability to handle comparisons between signed numbers.

In practical computing circuits, employing the **two's complement representation** offers convenience for representing negative numbers during addition and subtraction operations. In this coding scheme, the two's complement of a positive number remains unchanged, whereas the two's complement of a negative number involves inverting the original code and adding 1 to the result. This method enables efficient arithmetic operations for both positive and negative values within computing circuits.

(1) Subtraction is expressed using a full adder

I just mentioned the two's complement representation of negative numbers. In circuit design, negating can be represented by a **NOT gate**, while adding 1 can be represented by **setting the carry of the zeroth bit to 1**, that means $c_0 = 1$.

(2) Denote the case $\boldsymbol{A = B}$ $(\boldsymbol{AeqB})$

As in Experiment 1, the equality of two numbers can be represented by XNOR. Let $i_k = \overline{a_k \oplus b_k}$, so *AeqB* can be expressed as

$$\begin{aligned} AeqB &= i_3 i_2 i_1 i_0 \\ &= \left(\overline{a_3 \oplus b_3}\right).\left(\overline{a_2 \oplus b_2}\right).\left(\overline{a_1 \oplus b_1}\right).\left(\overline{a_0 \oplus b_0}\right) \\ &= \overline{\overline{\left(\overline{a_3 \oplus b_3}\right).\left(\overline{a_2 \oplus b_2}\right).\left(\overline{a_1 \oplus b_1}\right).\left(\overline{a_0 \oplus b_0}\right)}} \\ &= \overline{(a_3 \oplus b_3) + (a_2 \oplus b_2) + (a_1 \oplus b_1) + (a_0 \oplus b_0)} \end{aligned}$$

In cascaded multiple full adders circuit, due to b_i negated, $s_i = a_i \oplus \overline{b}_i \oplus c_{i-1}$, let $c_{i-1} = 1$ is a constant. Therefore, $s_i = \overline{a_i \oplus \overline{b}_i} = a_i \oplus b_i$. In order to satisfy *AeqB*, the logical expression is set to

$$\begin{aligned} Z &= \overline{(a_3 \oplus b_3) + (a_2 \oplus b_2) + (a_1 \oplus b_1) + (a_0 \oplus b_0)} \\ &= \overline{s_3 + s_2 + s_1 + s_0} \end{aligned}$$

where ‘Z’ means zero, if $A = B, Z = 1$.

(3) Denote the case $\boldsymbol{A > B(AgtB)\ and\ A < B(AltB)}$

In two's complement codes, the most significant bit signifies the sign of a number. Consequently, after completing the addition or summation operation, if the most significant bit $s_3 = 1$, it indicates that the sum is greater than 0, implying that $A > B$ (AgtB). Conversely, if the most significant bit $s_3 = 0$, it signifies that the sum is less than 0, indicating that $A < B$ (AltB).

Therefore, let $\boldsymbol{N = s_3}$, where ‘N’ means negative. If $N = 0, A < B$, and if $N = 1, A > B$.

(4) Denote that A and B are different sign.

If A and B are different signs, then just using the most significant bit to indicate positive or negative is not correct. At this point, it is necessary to consider whether the sign overflow.

For the two's complement code of signed numbers, the overflow generation is only related to the most significant bit c_4 and the second most significant bit c_3 when performing addition or subtraction operations. The logical expression for overflow is

$$V = c_4 \oplus c_3$$

Considering the above four cases, the design of the circuit diagram is shown in figure 5.

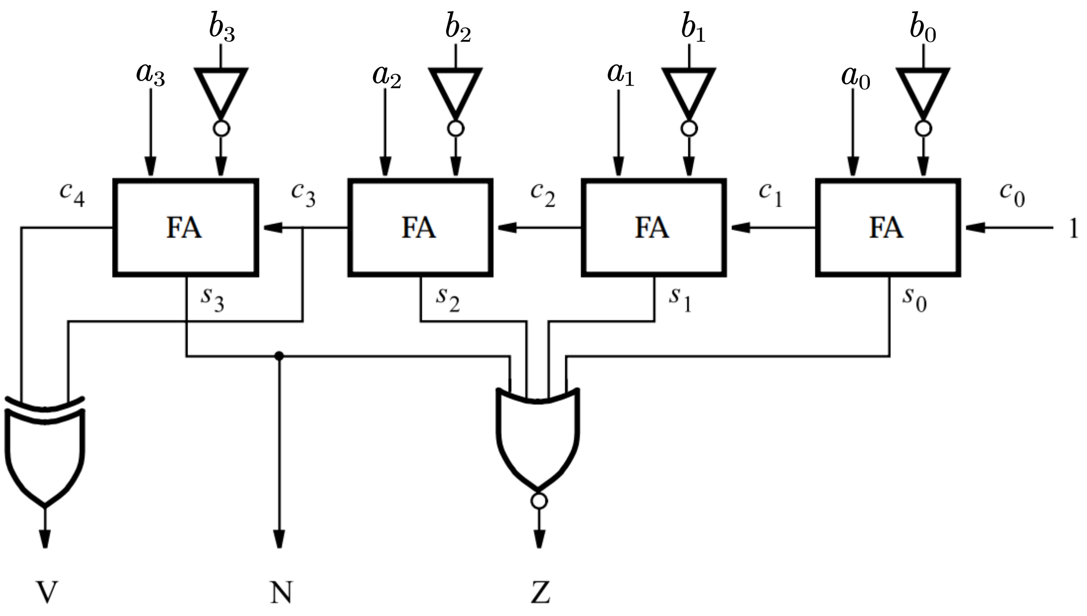


Figure 5: Designed circuit diagram of 4-bit comparator

3.4. Circuit analysis

(1) Case 1: $A = B$

If $A = B$, $c_4c_3c_2c_1c_0 = 11111$, $s_3s_2s_1s_0 = 0000$. Therefore, $V = c_4 \oplus c_3 = 0$, $N = s_3 = 0$. $Z = \overline{s_3 + s_2 + s_1 + s_0} = 1$.

(2) Case 2: $A > B$

If A and B have the same sign there will be no overflow, hence $V = 0$. Then for both positive and negative A and B the difference will be positive ($N = 0$).

If A is positive and B is negative, the difference will be positive ($N = 0$) if there is no overflow ($V = 0$); but the results will be negative ($N = 1$) if there is overflow ($V = 1$). Therefore, **if $A > B$ then $\overline{N \oplus V} = 1$.**

$A \geq B$ is indicated by $\overline{Z + (N \oplus V)} = 1$

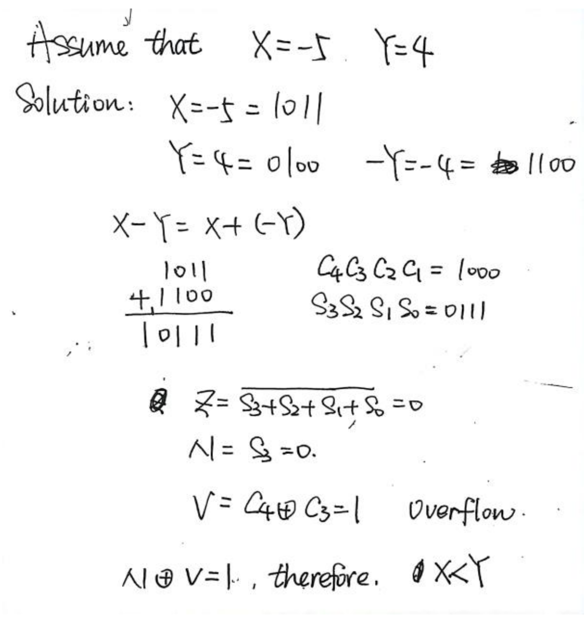
(3) Case 3: $A < B$

If A and B have the same sign there will be no overflow, hence $V = 0$. Then for both positive and negative A and B the difference will be negative ($N = 1$).

If A is negative and B is positive, the difference will be negative ($N = 1$) if there is no overflow ($V = 0$); but the results will be positive ($N = 0$) if there is overflow ($V = 1$). Therefore, **if $A < B$ then $N \oplus V = 1$**

$A \leq B$ is indicated by $Z + (N \oplus V) = 1$.

Figure 6 illustrates a situation that I assume to justify my corollary above.



Assume that $X=-5 \quad Y=4$

Solution: $X=-5=1011$

$Y=4=0100 \quad -Y=-4=1100$

$X-Y=X+(-Y)$

$$\begin{array}{r} 1011 \\ +\;1100 \\ \hline 10111 \end{array} \qquad \begin{array}{l} C_4C_3C_2C_1=1000 \\ S_3S_2S_1S_0=0111 \end{array}$$

$Z=\overline{S_3+S_2+S_1+S_0}=0$

$N=S_3=0.$

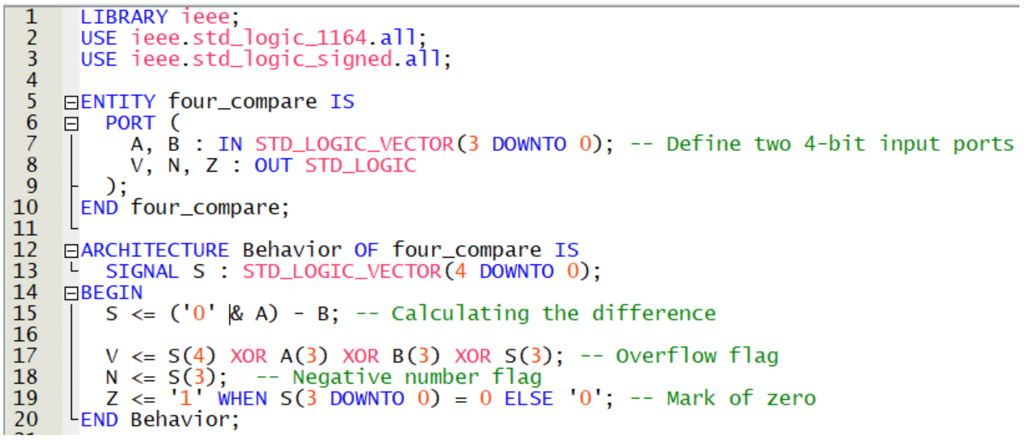
$V=C_4\oplus C_3=1$ Overflow.

$N\oplus V=1$, therefore, $X<Y$

Figure 6

3.5. Simulation and Results analysis

Using CAD tools, I wrote a VHDL code based on the logical expression of Z, V, N to simulate the above digital system design. Figure 7 shows my code section.



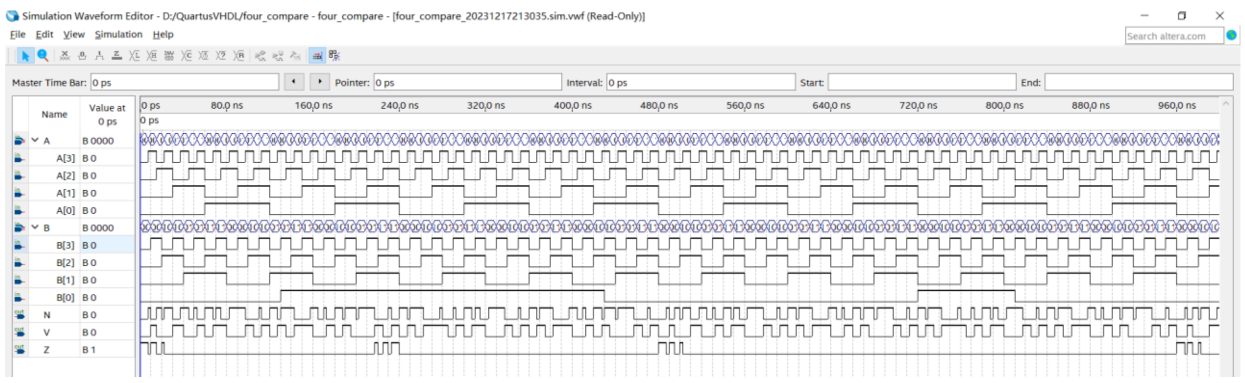
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY four_compare IS
  PORT (
    A, B : IN STD_LOGIC_VECTOR(3 DOWNTO 0); -- Define two 4-bit input ports
    V, N, Z : OUT STD_LOGIC
  );
END four_compare;

ARCHITECTURE Behavior OF four_compare IS
  SIGNAL S : STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
  S <= ('0' & A) - B; -- Calculating the difference

  V <= S(4) XOR A(3) XOR B(3) XOR S(3); -- Overflow flag
  N <= S(3);   -- Negative number flag
  Z <= '1' WHEN S(3 DOWNTO 0) = 0 ELSE '0'; -- Mark of zero
END Behavior;
```

Figure 7: Code section



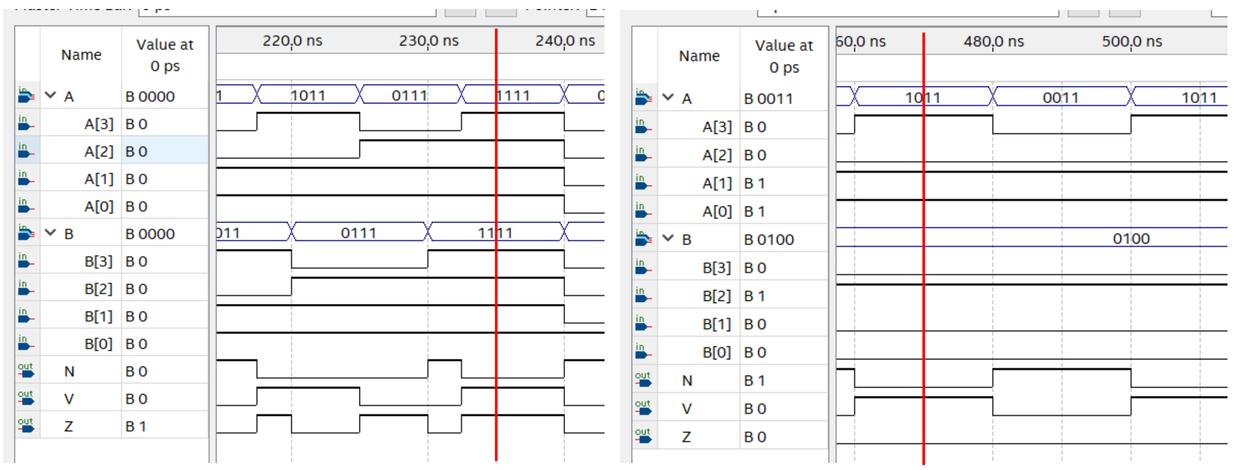


Figure 8: Simulation results

Figure 8 shows the corresponding simulation results. I took two sets of values for A and B, respectively, to verify that my results were correct.

For $A = 1111, B = 1111$. It can be seen that $N = 0, V = 1, Z = 1$. Since $Z = 1$, the 2 number are equal.

For $A = 1011, B = 0100$. These two numbers are the ones I used in figure 6 to verify that my theoretical inferences are correct. It can be seen that $N = 0, V = 1, Z = 0$, $N \oplus V = 1$. Therefore, $A < B$. This is consistent with my previous derivation in figure 6.

According to the two results selected above, it can be inferred that my simulation is correct.

4. Conclusion

In this digital system design and simulation experiment, I tackled the creation and simulation of both 2-bit and 4-bit comparators. For the 2-bit comparator, I employed a truth table to deduce the respective logical expression. However, for the 4-bit comparator, I utilized a cascaded full adder configuration. One of the significant advantages of this approach is its scalability to an n-bit comparator circuit, allowing for seamless extension and application to larger bit sizes.

Through engaging in this digital system design process, I gained insights into the underlying design principles of this discipline. Additionally, I learned how to leverage Computer-Aided Design (CAD) tools for simulation purposes. These tools played a pivotal role in modeling, analyzing, and validating the functionality of the designed digital systems. This experience provided a comprehensive understanding of design concepts and the practical utilization of CAD tools for simulating and verifying complex digital circuits.

References

[1] Jiang Jiaming, Lin Xia. Design and Functional Simulation of Binary Subtractor [J]. Fujian Computer,2022,38(06):83-86.

[2] Vandana Choudhary, Rajesh Mehra. 2-Bit Comparator Using Different Logic Style of Full Adder[J]. International Journal of Soft Computing and Engineering, 2013, 3(2): 1-124.
[3] Brown S, Vranesic Z. Foundations of Digital Logic and VHDL Design (3rd edition) [M]. McGraw-Hill Publishing, 2009.

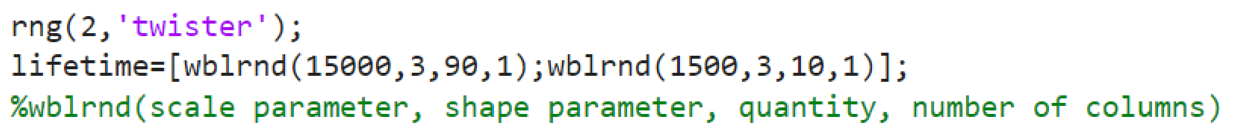
<u>Lab 2: Analyzing Survival and Reliability Data</u>

1. Introduction

This report presents the methodology and outcomes of the analysis conducted on survival and reliability data. Through the utilization of MATLAB code, this report demonstrates the creation of functions, curve fitting, and subsequent data analysis. The primary objective of this report is to comprehend fundamental concepts in reliability engineering, recognize the significance of selecting appropriate probability density functions for modeling reliability data, and ultimately, predict and analyze equipment reliability using life data.

2. Example: Analyze lifetime data with censoring

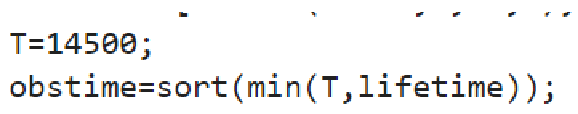
In this case, I analyzed the time to failure of an industrial power plant boiler system. Start by entering the following command into MATLAB to simulate the results of testing 100 controllers.



```
rng(2,'twister');
lifetime=[wblrnd(15000,3,90,1);wblrnd(1500,3,10,1)];
%wblrnd(scale parameter, shape parameter, quantity, number of columns)
```

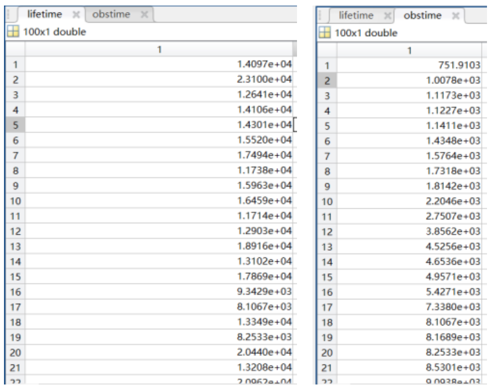
Two sets of random samples following the Weibull distribution are generated and combined into a matrix ‘lifetime’, with 90 samples in the first column and 10 samples in the second column. These samples simulate the lifetime distribution of the product or system under test, at the time points that may occur during the stress test. The resulting 'lifetime' is a 100×1 matrix.

The controller undergoes rigorous testing under high-pressure conditions, with each hour of testing deemed equivalent to 100 hours of real-world usage in the field. In this simulated environment, a total test duration of 14,500 hours has been selected to assess its performance. As a standard practice in simulated data analysis, the failure times are organized in ascending order to facilitate accurate evaluation.



```
T=14500;
obstime=sort(min(T,lifetime));
```

These two lines define the test times and sort the failure times using 'sort'. Figure 9 compares the data before ("lifetime") and after (" obstime") sorting.



lifetime | obstime

100x1 double

	1
1	1.4097e+04
2	2.3100e+04
3	1.2641e+04
4	1.4106e+04
5	1.4301e+04
6	1.5520e+04
7	1.7494e+04
8	1.1738e+04
9	1.5963e+04
10	1.6459e+04
11	1.1714e+04
12	1.2903e+04
13	1.8916e+04
14	1.3102e+04
15	1.7869e+04
16	9.3429e+03
17	8.1067e+03
18	1.3349e+04
19	8.2533e+03
20	2.0440e+04
21	1.3208e+04

lifetime | obstime

100x1 double

	1
1	751.9103
2	1.0078e+03
3	1.1173e+03
4	1.1227e+03
5	1.1411e+03
6	1.4348e+03
7	1.5764e+03
8	1.7318e+03
9	1.8142e+03
10	2.2046e+03
11	2.7507e+03
12	3.8562e+03
13	4.5256e+03
14	4.6536e+03
15	4.9571e+03
16	5.4271e+03
17	7.3380e+03
18	8.1067e+03
19	8.1689e+03
20	8.2533e+03
21	8.5301e+03

Figure 9

Finally, show the observation number versus the survival time of controllers, set T=14500 as the censored time, that is, test up to 14500 hours.

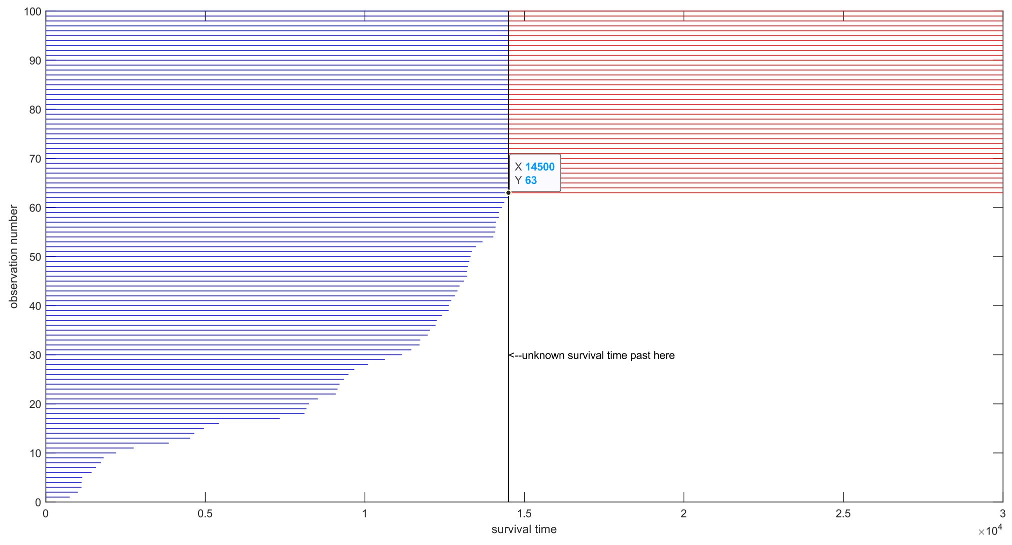


Figure 10: Relationship between survival time and observation number

Figure 10 shows the results, **37%** of the data are censored at 14,500 hours. The censored part represents the number of systems that were still alive or did not fail during the observation time, and these systems are treated as censored data. Similarly, it can be observed that at the end of the test, 38 of the 100 controllers out of 62 were damaged and survived.

3. Task 1: Examining the data distributions

In this task, the initial step involves generating a vector comprising 40,000 data points that are evenly spaced a linearly spaced vector. Following that, the Probability Density Function (PDF) of the Weibull distribution, along with the reliability or survival function, hazard rate, and Cumulative Distribution Function (CDF), will be plotted. This visualization can be accomplished using probability plots or rescaled graphs to illustrate these statistical functions derived from the Weibull distribution.

3.1. Establishment of mathematical model

(1) The probability density function (PDF) of Weibull distribution

$$f(t) = \beta\eta^{-\beta}(t-\gamma)^{\beta-1}e^{-\left(\frac{t-\gamma}{\eta}\right)^{\beta}} \tag{1}$$

where 't' means time, 'β' means shape parameter, 'γ' is the location or time delay parameter, 'η' is the scale parameter.

Equation (1) shows the PDF of Weibull distribution. This equation describes the probability density function of the Weibull distribution, which is used to describe the probability density of the occurrence of events of a random variable at different points in time. It is used in reliability analysis to study the lifetime distribution of a product or system, as well as to model and predict the probability of event occurrence.

(2) Reliability/survival function

The reliability function is:

$$R(t) = \int_t^{\infty} f(\tau)d\tau = e^{-\left(\frac{t-\gamma}{\eta}\right)^{\beta}} \tag{2}$$

The cumulative distribution function is:

$$F(t) = \int_{-\infty}^{t} f(\tau)d\tau$$

Therefore, the reliability equation can be written as:

$$R(t) = 1 - F(t) \tag{3}$$

(3) The hazard rate

$$h(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{1-F(t)} = \frac{\beta}{\eta}\left(\frac{t-\gamma}{\eta}\right)^{\beta-1} \tag{4}$$

$h(t)$ is the risk function and represents the rate or risk of failure of the system at time t at that instant. This function represents the probability density of a failure per unit time at time t.

(4) A probability plot for function 1 only with 50 data points

Probability plots: Probability plots serve as a valuable tool to contrast observed data against theoretical distributions. This method involves plotting the cumulative probability of observed data points on a theoretical distribution, juxtaposed with the corresponding cumulative probability on the anticipated distribution. When examining a probability plot, if the observed data align well with a theoretical distribution model, the data points are expected to roughly conform to a straight line, indicating a closer fit between the observed and expected distributions.

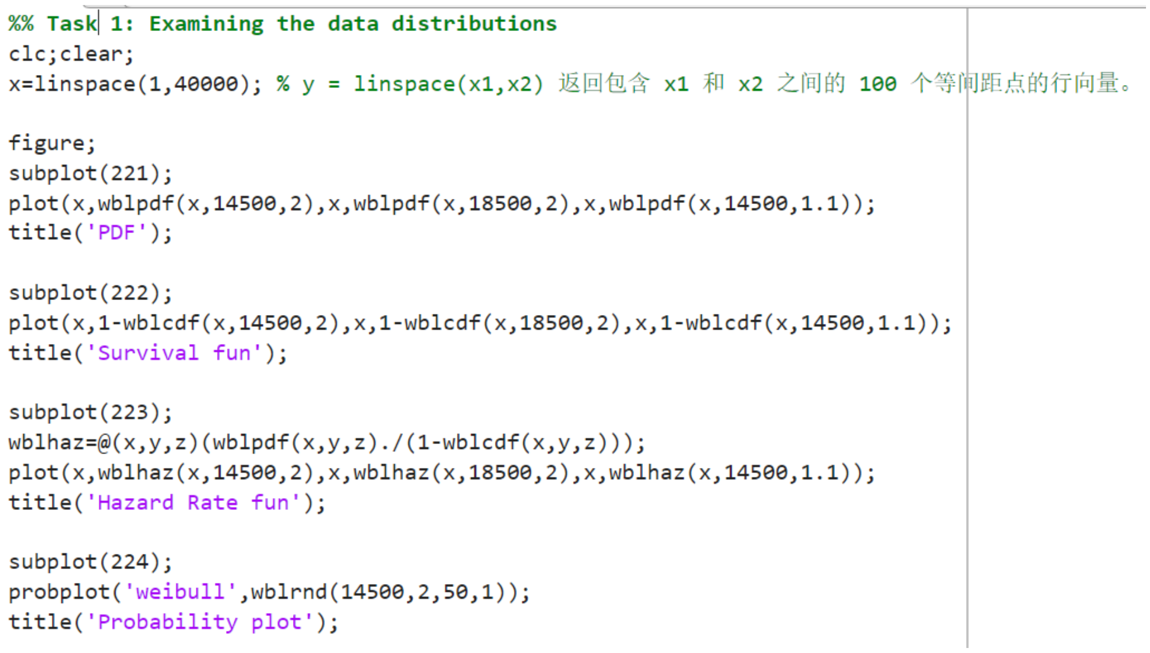
In this task, we select Weibull distribution as the distribution model to be fitted, and draw its probability plot, which can be used to visually evaluate the distribution of data and the degree of fit between the selected distribution model.

3.2. Solution of the Model

Table 3: Parameters of three Weibull distributions

	Scale parameter η	**Shape parameter β**
Distribution 1	14500	2
Distribution 2	18500	2
Distribution 3	14500	1.1

Firstly, I use “linspace()” function to generate 100 data points between 1~40,000. For PDF, I use “wblpdf” function to achieve the requirements. For reliability/survival function, I use “1-wblcdf” to achieve the requirements. For hazard rate, I create a new function “wblhaz” to achieve the requirements. For probability plot, I use “wblrnd” function to generate 50 data points, then use “probplot” function to fit the curve.



```
%% Task 1: Examining the data distributions
clc;clear;
x=linspace(1,40000); % y = linspace(x1,x2) 返回包含 x1 和 x2 之间的 100 个等间距点的行向量。

figure;
subplot(221);
plot(x,wblpdf(x,14500,2),x,wblpdf(x,18500,2),x,wblpdf(x,14500,1.1));
title('PDF');

subplot(222);
plot(x,1-wblcdf(x,14500,2),x,1-wblcdf(x,18500,2),x,1-wblcdf(x,14500,1.1));
title('Survival fun');

subplot(223);
wblhaz=@(x,y,z)(wblpdf(x,y,z)./(1-wblcdf(x,y,z)));
plot(x,wblhaz(x,14500,2),x,wblhaz(x,18500,2),x,wblhaz(x,14500,1.1));
title('Hazard Rate fun');

subplot(224);
probplot('weibull',wblrnd(14500,2,50,1));
title('Probability plot');
```

Figure 11: Code section

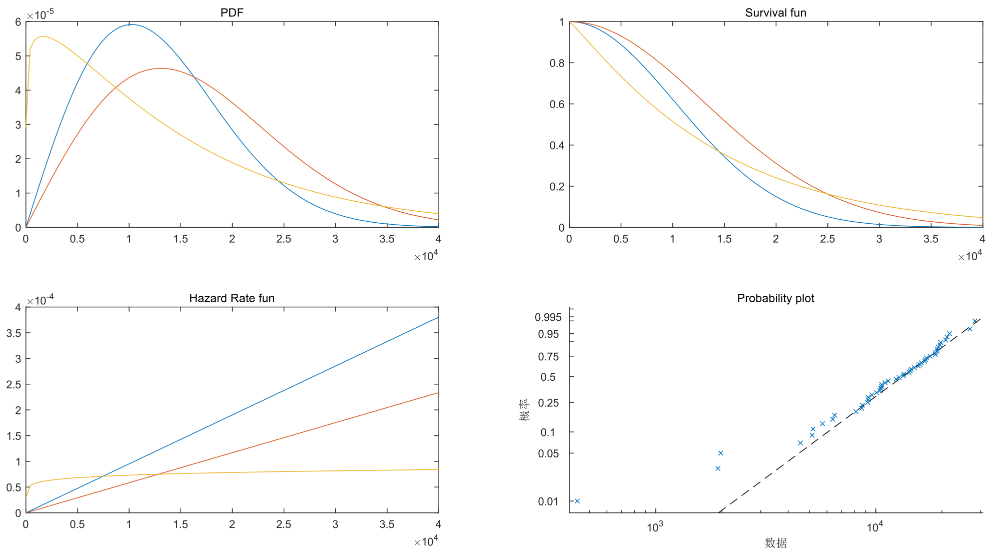


Figure 12: Results

Figure 11 shows the codes of this task, figure 12 shows the corresponding results. The blue line is distribution 1, the red line is distribution 2, the yellow line is distribution 3.

3.3. Results analysis

Combining Figure 12 with the theory learned in class, I can draw some conclusions:

(1) For the probability density function (PDF) of Weibull distribution, different parameters will affect its shape and characteristics.

The scale parameter η affects the scale of the distribution, that is, the stretching or compression of the probability density function on the horizontal axis. As η increases, the distribution becomes flatter, shifting to the right with respect to the origin of the coordinates; As η decreases, the distribution is steeper, shifting to the left with respect to the origin of the coordinates.

The shape parameter β controls the shape of the Weibull distribution. When β increases, the curve is more symmetric and closer to the normal distribution. When β is small, the curve may be more skewed or asymmetric.

(2) The shape and scale parameters likewise have a significant impact on the reliability equation.

The scale parameter η affects the rate at which the system failure rate changes over time. A large value of η corresponds to a slow change of system failure rate with time, that is, a long system life. However, a smaller value of η corresponds to a faster change in system failure rate with time and a shorter system life.

The shape parameter β affects the failure rate of the system. When β increases, the failure rate also increases, that is, the system is more likely to fail at earlier time points. Therefore, a larger value of β corresponds to a higher failure rate.

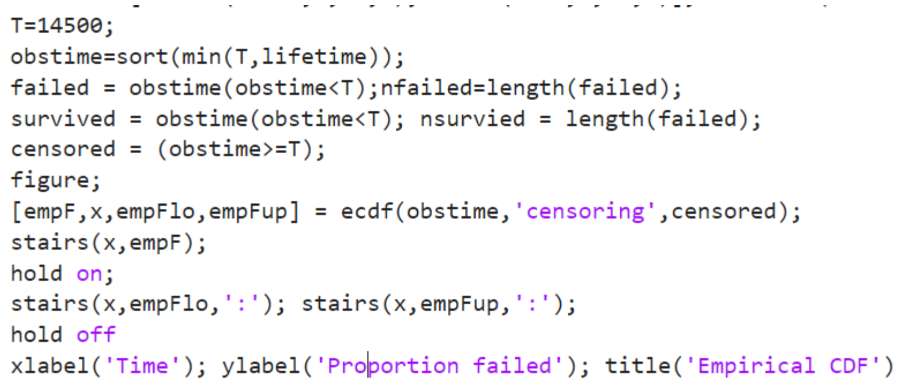
(3) The most significant influence on the hazard function is the shape parameter, where η controls the magnitude of the slope, which is a monotonically decreasing function when $\eta < 1$ and a monotonically increasing function when $\eta > 1$. The larger η, the larger the slope

4. Task 2: Fitting a Weibull distributions

4.1. Process of experiment

4.1.1.

Firstly, plot the empirical CDF of the lifetime data to show the proportion failing up to each possible survival time. The empirical CDF can be implemented using the code in the following figure. In these code, I use "ecdf()" function to generate CDF and its corresponding confidence interval and use "stairs" to plot a Stairstep graph.



```
T=14500;
obstime=sort(min(T,lifetime));
failed = obstime(obstime<T);nfailed=length(failed);
survived = obstime(obstime<T); nsurvied = length(failed);
censored = (obstime>=T);
figure;
[empF,x,empFlo,empFup] = ecdf(obstime,'censoring',censored);
stairs(x,empF);
hold on;
stairs(x,empFlo,':'); stairs(x,empFup,':');
hold off
xlabel('Time'); ylabel('Proportion failed'); title('Empirical CDF')
```

Figure 13: Code section

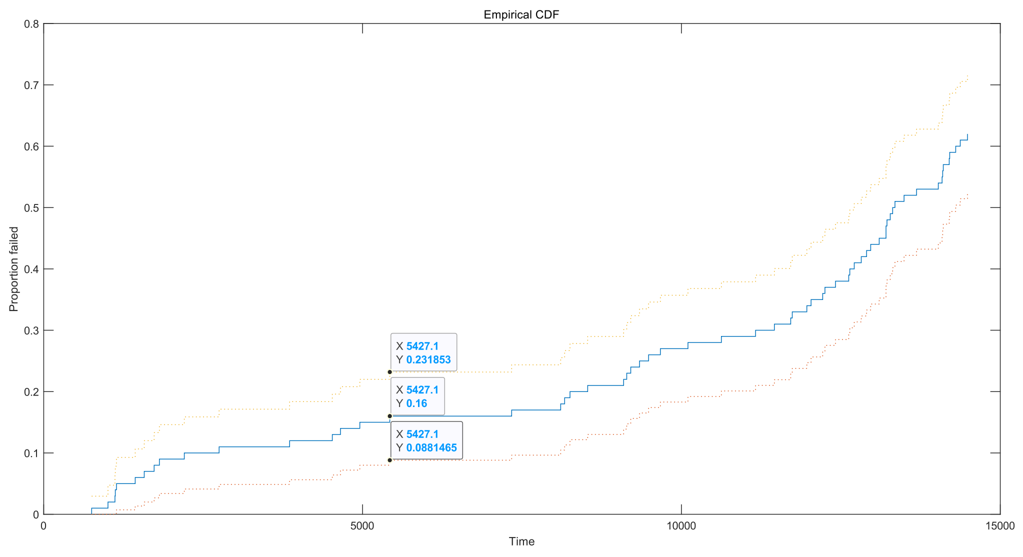


Figure 14: Empirical CDF results

As shown in figure 14, I picked $t = 5427$ and found that it failed 16% of the time, with an allowable failure rate of **8% to 23%** at its 95% confidence interval.

- **Explication:**

(1) Empirical CDF

For a sequence of samples $\{X_i\}_{i=1}^{n}$, the empirical CDF can be defined as

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} 1\{X_i \leqslant x\} \quad (x \in \mathbb{R})$$

where $1\{X_i \leqslant x\}$ is an indicator function, if $X_i \leqslant x$, the indicator function is set to 1, otherwise set to 0. Therefore, F_n reflects the fraction of elements in the sample that are less than x.

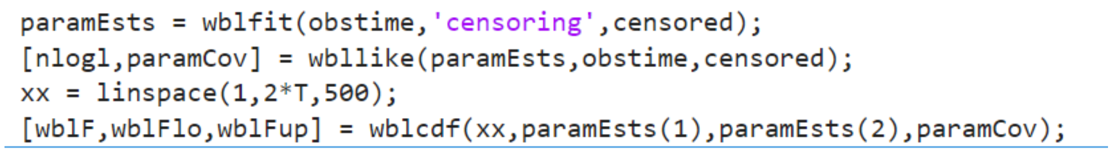
The empirical CDF shows the proportion of failures at each possible lifetime value in the observed data. Through the Stairstep graph, it can be clearly seen that the proportion of failure is gradually increasing as time goes by.

(2) Censoring impact on the empirical CDF

Censoring's impact on the empirical Cumulative Distribution Function (CDF) manifests as the potential truncation or absence of certain observations. Consequently, sections of the data might be incomplete or unobservable along the timeline, resulting in partially missing information displayed within the CDF rather than accurately observed data. This leads to a representation where certain segments of the timeline lack precise observations due to the censoring effect, creating gaps or inaccuracies in the CDF depiction.

4.1.2.

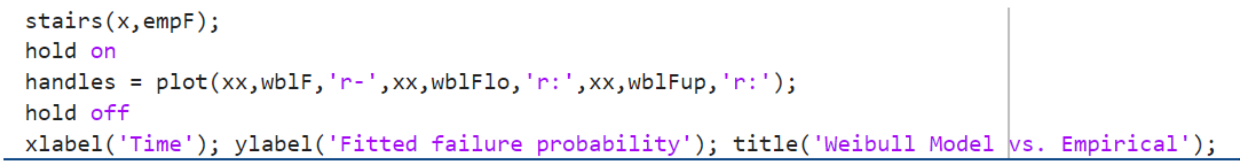
Fit the Weibull distribution to the data, including censored data, using the "wblfit" function. After calculating the parameter estimates, these estimates are used to evaluate the CDF of the fitted Weibull model.



```
paramEsts = wblfit(obstime,'censoring',censored);
[nlogl,paramCov] = wbllike(paramEsts,obstime,censored);
xx = linspace(1,2*T,500);
[wblF,wblFlo,wblFup] = wblcdf(xx,paramEsts(1),paramEsts(2),paramCov);
```

4.1.3.

The plots of empirical CDF and fitted CDF are superimposed to judge the modeling effect of Weibull distribution on reliability data.



```
stairs(x,empF);
hold on
handles = plot(xx,wblF,'r-',xx,wblFlo,'r:',xx,wblFup,'r:');
hold off
xlabel('Time'); ylabel('Fitted failure probability'); title('Weibull Model vs. Empirical');
```

Figure 15: Code section

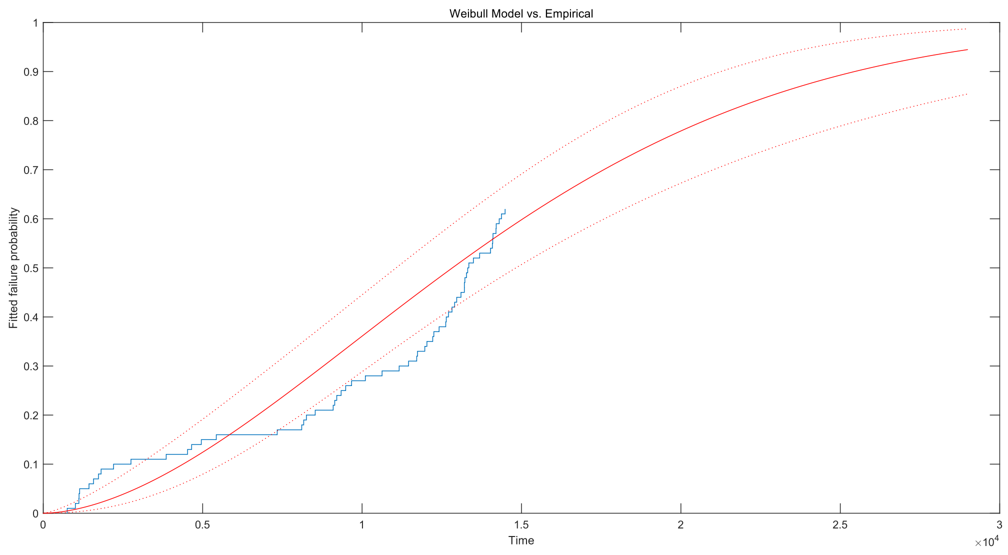


Figure 16: Results of empirical CDF and the fitted CDF

4.2. Results analysis

Figure 16 exhibits a juxtaposed plot of the empirical and fitted Cumulative Distribution Functions (CDFs). The red line represents the fitted CDF, while the blue line represents the empirical CDF. Additionally, two red dashed lines are employed to demarcate the confidence intervals. Notably, the Weibull distribution frequently serves as a reliable model for equipment failure.

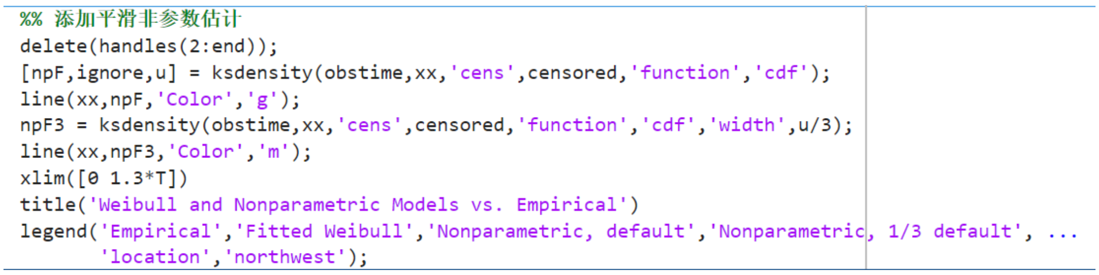
The main difference between the empirical cumulative distribution function (ECDF) and the fitted CDF is that the ECDF is derived from the actual observed data, which directly shows the distribution of the data. The fitting CDF is based on a specific probability distribution model and tries to find the theoretical probability distribution that best fits the data by fitting the data with a mathematical function.

The Weibull model allows us to predict and compute failure probabilities beyond the end of the testing period. However, it is observed that the fitted curve does not match our data well. There are more early failures observed before the time point of 2,000 compared to what the Weibull model predicts, and there are fewer failures between approximately 7,000 and 13,000. Therefore, **the fitted function does not represent the empirical data well**.

4.3. Improve fitting the CDF

Even the Weibull distribution does not fit the curve well, so the method of non-parametric estimation is considered. Adding **a smooth nonparametric estimation** as a new method.

Unlike parametric estimators, non-parametric estimators do not need to impose any parametric assumptions, so they can avoid making significant errors due to improper assumptions about the population distribution, and so are often more robust.



```matlab
%% 添加平滑非参数估计
delete(handles(2:end));
[npF,ignore,u] = ksdensity(obstime,xx,'cens',censored,'function','cdf');
line(xx,npF,'Color','g');
npF3 = ksdensity(obstime,xx,'cens',censored,'function','cdf','width',u/3);
line(xx,npF3,'Color','m');
xlim([0 1.3*T])
title('Weibull and Nonparametric Models vs. Empirical')
legend('Empirical','Fitted Weibull','Nonparametric, default','Nonparametric, 1/3 default', ...
       'location','northwest');
```

Figure 17: Code section

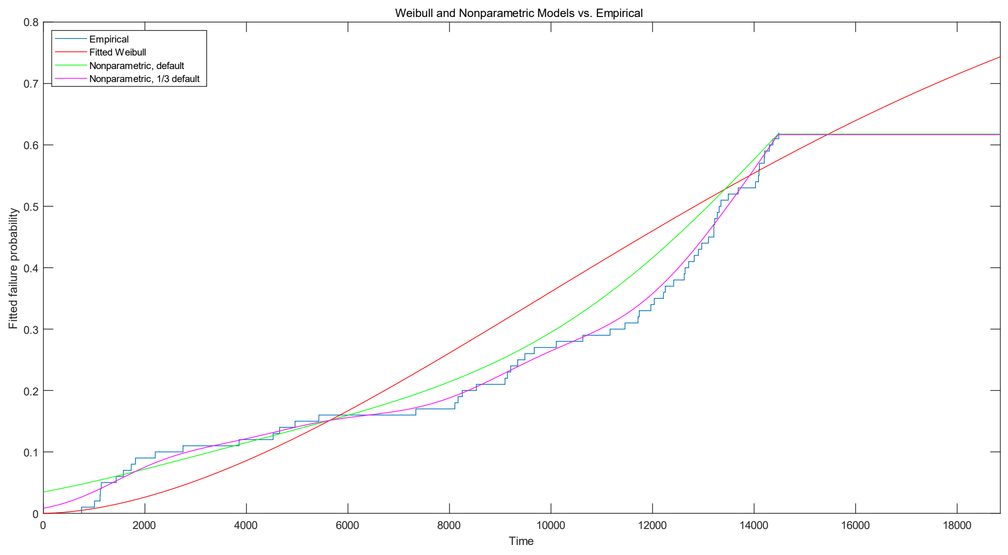


Figure 18: Non-parametric estimation results

Figure 17 shows the corresponding code. In this code, I conduct two nonparametric estimates utilizing the "ksdensity" function. One estimation uses the default smoothing parameter, while the other employs a smoothing parameter that is 1/3 of the default value. The purpose is to juxtapose these nonparametric estimations with the empirical data and the Weibull fitting model at distinct levels of smoothness.

Figure 18 shows the non-parametric estimation results. It can be found that **the smaller smoothing parameter makes the curve follow the data more closely**. Based on the observations, it is evident that the non-parametric estimation yields a notably superior fit compared to the parametric estimation. Consequently, **the nonparametric estimation optimizes the fit to the CDF**.

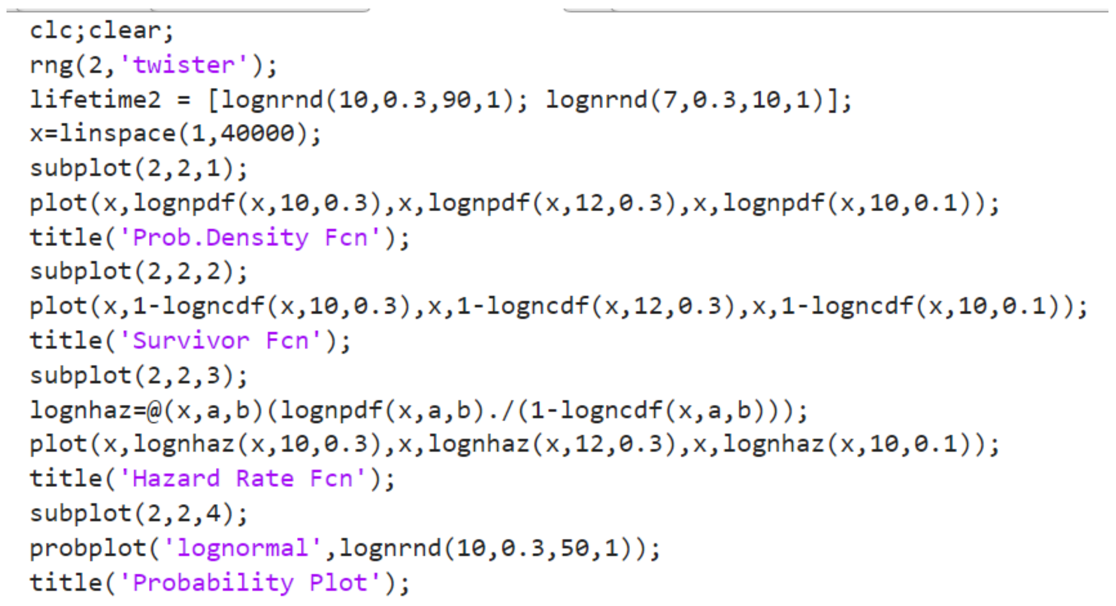
5. Further task

The lifetime of the controller modeled with a **lognormal distribution** repeats task 1, using the lifetime data in Table 4. Similarly, I need simulate PDF, reliability function, hazard rate and CDF with 50 data points.

Table 4: Parameters of 3 log-normal distributions

	Mean	**Variance**

Distribution 1	10	0.3
Distribution 2	12	0.3
Distribution 3	10	0.1



```
clc;clear;
rng(2,'twister');
lifetime2 = [lognrnd(10,0.3,90,1); lognrnd(7,0.3,10,1)];
x=linspace(1,40000);
subplot(2,2,1);
plot(x,lognpdf(x,10,0.3),x,lognpdf(x,12,0.3),x,lognpdf(x,10,0.1));
title('Prob.Density Fcn');
subplot(2,2,2);
plot(x,1-logncdf(x,10,0.3),x,1-logncdf(x,12,0.3),x,1-logncdf(x,10,0.1));
title('Survivor Fcn');
subplot(2,2,3);
lognhaz=@(x,a,b)(lognpdf(x,a,b)./(1-logncdf(x,a,b)));
plot(x,lognhaz(x,10,0.3),x,lognhaz(x,12,0.3),x,lognhaz(x,10,0.1));
title('Hazard Rate Fcn');
subplot(2,2,4);
probplot('lognormal',lognrnd(10,0.3,50,1));
title('Probability Plot');
```

Figure 19: Code section

In code section, I use “lifetime2 = [lognrnd(10,0.3,90,1); lognrnd(7,0.3,10,1)]” to generate two sets of log-normally distributed simulated data and combine them into one dataset. These data were simulated using the “lognrnd” function with two log-normal distributions with different parameters.

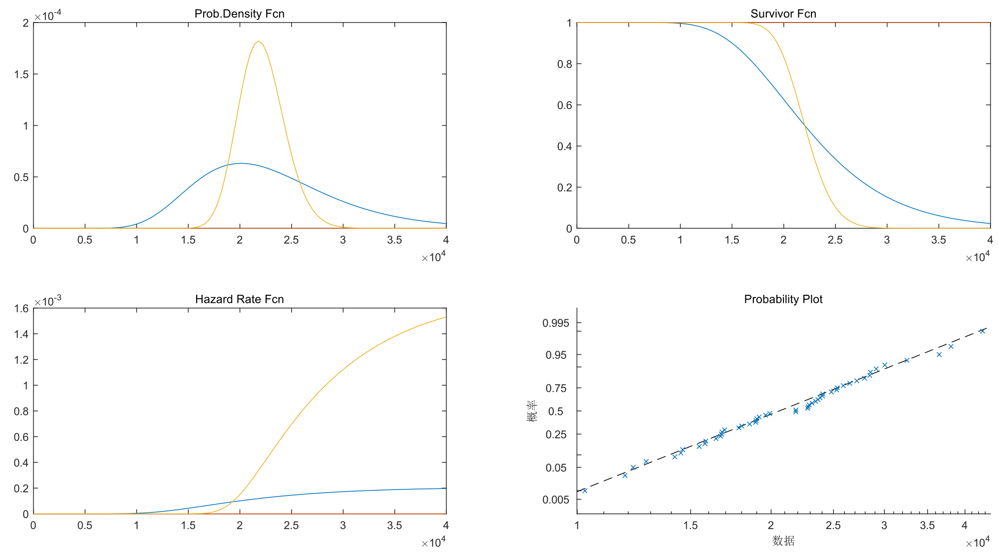


Figure 20: Log-normal distribution results

Figure 20 shows the results, where blue line represents distribution 1, red line represents distribution 2, yellow line represents distribution 3.

Similarly, use log-normal distribution repeats task 2. The code section and corresponding results is shown in figure 21 and figure 22, respectively.

```
%% repeat task 2
rng(2,'twister');
lifetime = [lognrnd(10,0.3,90,1); lognrnd(7,0.3,10,1)];
T=14500;
obstime=sort(min(T,lifetime));
failed = obstime(obstime<T);nfailed=length(failed);
survived = obstime(obstime<T); nsurvied = length(failed);
censored = (obstime>=T);
[empF,x,empFlo,empFup] = ecdf(obstime,'censoring',censored);

paramEsts = lognfit(obstime,'censoring',censored);
[nlogl,paramCov] = lognlike(paramEsts,obstime,censored);
x2=linspace(1,2*T,500);
[wblF,wblFlo,wblFup] =logncdf(x2,paramEsts(1),paramEsts(2),paramCov);
figure;
stairs(x,empF);
hold on
handles = plot(x2,wblF,'r-',x2,wblFlo,'r:',x2,wblFup,'r:');
hold off
xlabel('Time');ylabel('Fitted failure probability');
title('Weiibull Model vs. Empirical');
```

Figure 21: Code section

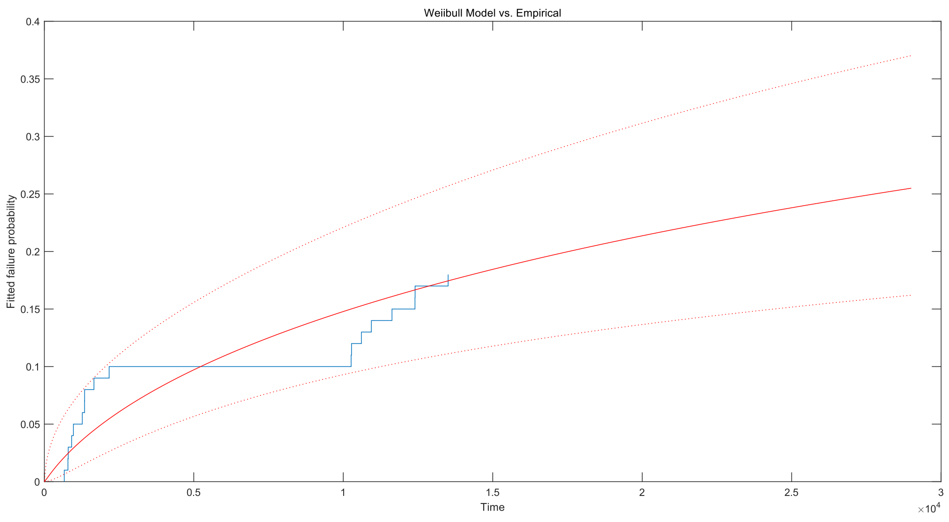


Figure 22: Log-normal distribution results

5.1. Results analysis

Observing the resulting plot, the following conclusions can be drawn:

(1) **Probability density function:** Mean influences the position, while the standard deviation affects the width of the distribution.

(2) **Reliability function**: Mean dictates the starting point, and standard deviation determines the rate of decay towards zero.

(3) **Hazard rate**: Mean influences the peak position, and standard deviation affects the rate of convergence towards zero.

(4) **Probability plot**: Larger mean and standard deviation may cause data points to deviate from the ideal line in the probability plot.

6. Conclusions

Through this experiment, I plotted the Probability Density Function (PDF), reliability equation, hazard rate, and Cumulative Distribution Function (CDF) as functions. After analyzing the fitting of these functions, I arrived at the conclusion that the smooth nonparametric estimation offers a superior fit to the CDF.

Moreover, I employed the logarithmic method to visualize the function curves. This experiment has equipped me with various data analysis techniques and insights into optimizing curve fitting through both parametric and non-parametric estimations.

References

[1] K. C. Kapur and M. Pecht, *Reliability Engineering*, John Wiley, and Sons, 2014