南京航空航天大學
NANJING UNIVERSITY OF AERONAUTICS AND ASTRONAUTICS

软件可靠性课程实验报告

题　　目： **GO** 软件可靠性模型

院　　系： 计算机科学与技术学院/软件学院

专　　业： 软件工程

姓　　名： 陈梓鹏

学　　号： 162230217

2024 年 11 月 21 日

目录

1.引言

1.1 编写目的

随着软件规模的不断扩大，系统结构日益复杂，应用范围也在逐步拓展，软件危机依然是我们在软件开发和应用过程中面临的一项重大挑战。这一危机不仅影响着软件的质量和性能，还对社会、经济以及军事等各个领域的应用产生深远影响。因此，如何有效地解决软件危机，提升软件的质量和可靠性，已成为摆在我们面前的紧迫问题。加强软件工程管理，优化软件开发过程，已不再是可选项，而是迫切需要解决的核心任务，甚至是我们克服软件危机、确保软件可持续发展的必然要求。

在这一背景下，提高软件的可靠性变得尤为重要，特别是在军事领域，为部队提供可靠、稳定且高效的装备，不仅是技术发展的需求，更是我们义不容辞的责任和使命。只有确保软件系统的高度可靠，才能在关键时刻发挥出最大的效能，保障部队的作战能力和安全。

本次试验以 GO 模型为核心，通过深入的分析和实践，旨在帮助我们全面理解软件可靠性评估的基本原理，掌握 GO 模型在实际中的应用方法。通过这一模型的学习与应用，我们可以更好地评估软件的可靠性，预测潜在的风险和问题，进而采取有效的措施加以改进，提升软件产品的整体质量和稳定性，确保它们能够在复杂和严苛的环境中持续发挥作用。

2.模型理论

2.1 模型背景

Goel-Okumto 软件可靠性模型（GO 模型）于 1979 年由 Geol 和 Okumoto 提出，属于 NHPP 有限错误模型。

2.2 模型假设

1) 程序在同实际执行环境相差不大的条件下执行。
2) 在软件测试过程中，所有检测到的故障是相互独立的。
3) 测试未运行时的软件失效为 0；当测试进行时，软件失效服从均值为 m（t）的非齐次泊松过程（NHPP）。
4) t 时刻累积检测到的故障数量 m（t）变化率与当前剩余的故障数量 a-m（t）成正比例，比例系数为 b。
5) 每次只修正一个错误，当软件故障出现时，引发故障的错误被立即排除，并不会引入新的错误

2.3 模型推导

GO 模型在测试区间[0,t]内的累计失效数期望函数为

$$m(t) = a(1 - e^{-bt})$$

t 为软件累计测试时间。

可靠性函数为：

$$R(s|t) = e^{a\left(e^{-b(t+s)} - e^{-bt}\right)} = e^{-m(s)e^{-bt}}$$

按假设，若 t 时刻累计故障数为 y，则得到 N(t)的概率密度为:

$$P_r\{N(t) = y\} = \frac{m(t)^y}{y!}e^{-m(t)} = \frac{(a(1 - e^{-bt}))^y}{y!}exp\{-a(1 - e^{-bt})\}$$

得出参数的似然函数为:

$$L(a,b) = \prod_{i=1}^{m} \frac{(a(e^{-bt_{i-1}} - e^{-bt_i}))^{n_i - n_{i-1}}}{(n_i - n_{i-1})!} \exp\{-a(1 - e^{-bt_m})\}$$

通过最大化似然函数来估计参数 a 和 b。

对于给定故障发生时间$s = (s_1, s_2)$

$$\frac{\partial \ln L(s_1, s_2, \ldots, s_N; a, b)}{\partial a} = \frac{\partial lnL(s_1, s_2, \ldots, s_N; a, b)}{\partial b} = 0$$

则有

$$\begin{cases} \frac{N}{\hat{a}} = 1 - e^{-\hat{b}s_N} \\ \frac{N}{\hat{b}} = \sum_{k=1}^{N} s_k + \hat{a}s_N e^{-\hat{b}s_N} \end{cases}$$

对该式进行求解，则可以得到$\hat{a}$,$\hat{b}$的值。

3.算法实现

3.1 算法原理

为了估计 GO 模型中的参数$\hat{a}$和$\hat{b}$，需要通过最大化似然函数来进行优化。这个过程通常通过数值优化方法来完成。

3.1.1 数值优化

数值优化是通过计算方法来寻找一个最优解（即最适合实际数据的模型参数）。在 G-O 模型中，我们需要找到能使得模型预测的故障数量和实际观测到的故障数量最接近的参数值。这个过程通常是通过最大化似然函数来实现的，似然函数衡量的是给定数据下，某一组参数的可能性有多大。

优化的过程本质上是通过调整参数$\hat{a}$和$\hat{b}$的值,逐步减小模型输出与实际数据之间的误差，直到找到使误差最小的参数值。

3.1.2 二分法的作用

二分法（也称为二分查找法）是一种简单且高效的数值优化方法，常用于寻找某个值的近似解。在 GO 模型中，二分法的应用步骤如下：

1) 首先，二分法会确定一个参数值的范围（例如$\hat{a}$或$\hat{b}$的可能范围）。
2) 然后，它通过计算中点x_m来评估当前范围内的最优解。
3) 通过比较当前计算结果与预期的精度目标（如误差小于某个阈值），二分法会根据计算结果调整参数的范围。
4) 每次迭代时，二分法都会将参数值的范围缩小一半，逐渐逼近最优的参数值。

具体而言，对于给定的误差值 D，二分法会选择一个初始的上下界（即x_l和x_r），并不断通过中间值x_m来调整参数，直到误差足够小，或者达到停止条件。

3.2 算法步骤

步骤 1：令给定的误差允许值为$D = \sum^N t_i/Nt_N$，如果 0<D<1/2，则$x_l = (1-2D)/2$，故$x_r = 1/D$，转步骤 2；如果 D>=1/2 则参数估计无解，转步骤 5

步骤 2：计算$x_m = (x_r + x_l)/2$，如果$|x_r\text{-}x_l| \leq \zeta_v$，则转步骤 4

步骤 3：$f = (1-Dx_m)e^{x_m} + (D-1)x_m - 1$，如果$f > \zeta_v$，则$x_l = x_m$转步骤 2；如果$f < -\zeta_v$，则$x_r = x_m$，转步骤 2

步骤 4：计算$b = x_m/t_N$和$a = N/(1-e^{-bt_N})$

步骤 5：停止计算

3.3 算法流程

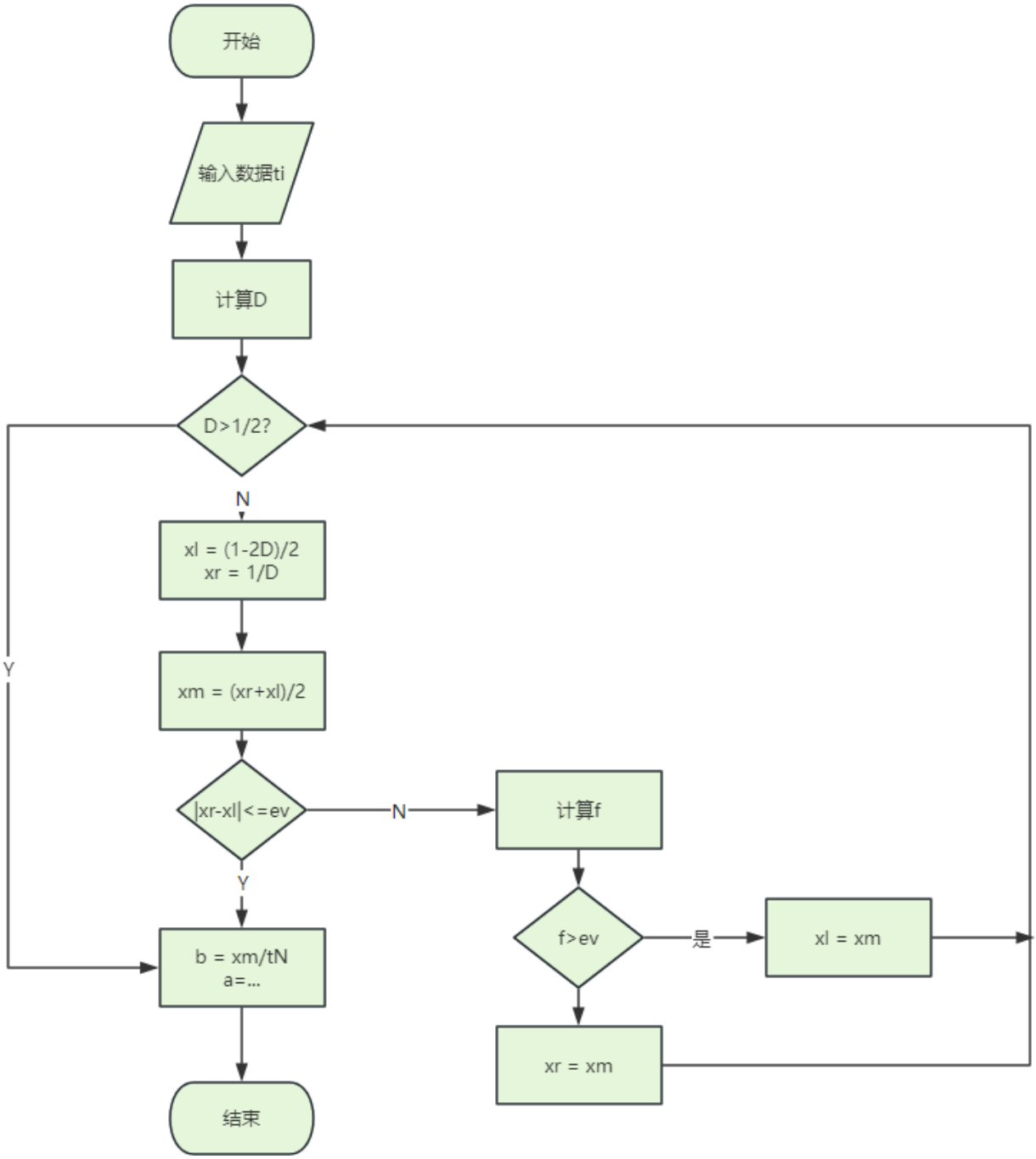

图 1.GO 模型算法流程图

3.4 算法伪代码

```
if (D >= 1 / 2)
    return;
else if (0 < D < 1 / 2)
```

```
{
    x_l = (1 - 2D) / 2;
    x_r = 1 / D;
}
while (| x_r - x_l | > ζ_v)
{
    x_m = (x_l + x_r) / 2;
    f = (1 - Dx_m) e ^ (x_m) + (D - 1) x_m - 1;
    if (f > ζ_v)
    {
        x_l = x_m;
        continue;
    }
    else if (f < -ζ_v)
    {
        x_r = x_m;
        continue;
    }
}
```

最后根据公式

$$b = \frac{x_m}{t_N}$$

$$a = \frac{N}{(1 - e^{-bt_N})}$$

既可得到 a,b 的参数估计值

3.5 算法实现

本次实验采用 Java 语言编写，运行的 IDE 是 vscode。

数据初始化：

```
    ArrayList<Double> T = new ArrayList<Double>(); // 存储失效时间
    double D = 0.0; // 参数D
    double f = 0.0; // 函数f 的结果
    double xl = 0.0; // 左边界
    double xr = 0.0; // 右边界
    double xm = 0.0; // 中点
    double b = 0.0; // 参数b
    double a = 0.0; // 参数a
    double v = 1.0E-1; // 精度控制
```

D 值计算：

```
// 计算D 值
    public double getD(G_O g_o) {
        double result = 0.0;
```

```
        double sum = 0.0;
        int n = g_o.T.size();

        // 累计失效时间求和
        for (int i = 0; i < n; ++i) {
            sum += g_o.T.get(i);
        }
        result = sum / (n * g_o.T.get(n - 1)); // 计算D
        return result;
    }
```

计算函数 f

```
// 计算函数f，用于判断中点xm是否满足条件
    public double getF(G_O g_o, double D) {
        double p1 = (1.0 - D * g_o.xm) * Math.pow(Math.E, g_o.xm);
        double p2 = (D - 1.0) * g_o.xm - 1.0;
        double result = p1 + p2;
        return result;
    }
```

计算参数 b

```
public double getb(G_O g_o) {
        int n = g_o.T.size();
        double result = g_o.xm / g_o.T.get(n - 1);
        return result;
    }
```

计算参数 a

```
public double geta(G_O g_o) {
        int n = g_o.T.size();
        double temp2 = -this.b * g_o.T.get(n - 1);
        double temp1 = Math.pow(Math.E, temp2);
        double result = n / (1.0 - temp1);
        return result;
    }
```

完整代码：

```
package softwarereality;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
public class G_O {
    ArrayList<Double> T = new ArrayList<Double>(); // 存储失效时间
    double D = 0.0; // 参数D
```

```
    double f = 0.0; // 函数f的结果
    double xl = 0.0; // 左边界
    double xr = 0.0; // 右边界
    double xm = 0.0; // 中点
    double b = 0.0; // 参数b
    double a = 0.0; // 参数a
    double v = 1.0E-4; // 精度控制
    public G_O() {
    }
    // 从文件中读取时间数据并填充到T列表中
    public void setT(G_O g_o) {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new
FileReader("D:\\javacode\\softwarereality\\go.txt"));
        } catch (FileNotFoundException var13) {
            var13.printStackTrace();
        }
        double temp = 0.0;
        String line = null;
        try {
            line = br.readLine();
        } catch (IOException var12) {
            var12.printStackTrace();
        }
        // 读取第一个时间数据
        String[] T0 = line.split("\\s+");
        temp = Double.valueOf(T0[1]);
        g_o.T.add(temp);
        try {
            line = br.readLine();
        } catch (IOException var11) {
            var11.printStackTrace();
        }
        // 读取剩余的数据
        while (line != null) {
            String[] numbers = line.split("\\s+");
            temp += Double.valueOf(numbers[1]); // 累计时间
            g_o.T.add(temp);
            try {
                line = br.readLine();
            } catch (IOException var10) {
                var10.printStackTrace();
            }
```

```
        }
        try {
            br.close();
        } catch (IOException var9) {
            var9.printStackTrace();
        }
    }
    // 计算D 值
    public double getD(G_O g_o) {
        double result = 0.0;
        double sum = 0.0;
        int n = g_o.T.size();
        // 累计失效时间求和
        for (int i = 0; i < n; ++i) {
            sum += g_o.T.get(i);
        }
        result = sum / (n * g_o.T.get(n - 1)); // 计算D
        return result;
    }
    // 计算函数f，用于判断中点 xm 是否满足条件
    public double getF(G_O g_o, double D) {
        double p1 = (1.0 - D * g_o.xm) * Math.pow(Math.E, g_o.xm);
        double p2 = (D - 1.0) * g_o.xm - 1.0;
        double result = p1 + p2;
        return result;
    }
    // 计算参数b
    public double getb(G_O g_o) {
        int n = g_o.T.size();
        double result = g_o.xm / g_o.T.get(n - 1);
        return result;
    }
    // 计算参数a
    public double geta(G_O g_o) {
        int n = g_o.T.size();
        double temp2 = -this.b * g_o.T.get(n - 1);
        double temp1 = Math.pow(Math.E, temp2);
        double result = n / (1.0 - temp1);
        return result;
    }
    public static void main(String[] args) {
        G_O g_o = new G_O();
        g_o.setT(g_o); // 读取失效时间数据
        g_o.D = g_o.getD(g_o); // 计算D 值
```

```
        // 如果D不在合适范围内，输出无解
        if (g_o.D >= 0.5) {
            System.out.println("参数估计无解" + g_o.D);
        } else {
            // 确定初始的左边界xl和右边界xr
            if (g_o.D > 0.0 && g_o.D < 0.5) {
                g_o.xl = (1.0 - 2.0 * g_o.D) / 2.0;
                g_o.xr = 1.0 / g_o.D;
            }
            // 二分法迭代逼近xm
            while (Math.abs(g_o.xr - g_o.xl) > g_o.v) {
                g_o.xm = (g_o.xr + g_o.xl) / 2.0;
                g_o.f = g_o.getF(g_o, g_o.D);
                if (g_o.f > g_o.v) {
                    g_o.xl = g_o.xm; // 更新左边界
                } else {
                    if (!(g_o.f < -g_o.v)) {
                        break; // 满足条件，退出迭代
                    }
                    g_o.xr = g_o.xm; // 更新右边界
                }
            }
            // 计算参数b和a
            g_o.b = g_o.getb(g_o);
            g_o.a = g_o.geta(g_o);
            System.out.println("b= " + g_o.b);
            System.out.println("a= " + g_o.a);
        }
    }
}
```

3.6 数据集

选取美国海军舰队计算机程序中心开发的海军战术数据系统收集的故障数据集 DSI，如表所示。

表 1.DSI 数据集

失效序号	失效时间	失效序号	失效时间	失效序号	失效时间	失效序号	失效时间
0	0	9	63	18	98	27	337
1	9	10	70	19	104	28	384
2	21	11	71	20	105	29	396
3	32	12	77	21	116	30	405
4	36	13	78	22	149	31	540
5	43	14	87	23	156	32	798
6	45	15	91	24	247	33	814
7	50	166	92	25	249	34	849
8	58	17	95	26	250		

3.7 算法结果

当ζ_v=0.1 时

```
b= 0.00442075031077805
a= 34.816167937815166
```

当ζ_v=0.01 时

```
b= 0.004411875172473548
a= 34.82249031627463
```

当ζ_v=0.001 时

```
b= 0.004406328211033234
a= 34.82646723910538
```

当ζ_v=0.0001 时

```
b= 0.004405842851907206
a= 34.826816155095166
```

4.JM 模型和 GO 模型比较

4.1 PLR 法

序列似然比是用来对两个模型预计质量的相对优劣进行评定的一个指标。通过计算这个指标，可以比较不同模型对某一分布所作预计的相对准确程度。

PLR 法的总体思路是认为软件失效发生在软件失效时间的分布函数密度大的地方的可能性较大。预计软件失效时间的分布函数密度大的预计相比较而言更接近于真实。

J-M 模型的预计的概率密度为：$f(x_i) = \psi(N_0 - i + 1)\, exp\{-\psi(N_0 - i + 1)x_i\}$

G-O 模型的预计的概率密度为：$P_r\{N(t_i) = i\} = \frac{(a(1-e^{-bt_i}))^i}{i!} exp\{-a(1-e^{-bt_i})\}$

$$PLR_{s,i}{}^{AB} = \prod_{j=s}^{j=i} \frac{\tilde{f^A}_j(t_j)}{\tilde{f^B}_j(t_j)}$$

$\widetilde{f_J^A}$表示预计模型 A 的预计概率密度，$\widetilde{f_J^B}$表示预计模型 B 的预计概率密度。如果$i \to \infty$，$PLR_{s,i}{}^{AB} \to \infty$，可以认为预计模型 A 优于预计模型 B，反之，可以认为预计模型 B 优于预计模型 A。如果比值既不趋于无穷又不趋于 0，而是趋近于某一个常数 c，则说明预计模型 A 和预计模型 B 提供不相上下的预计。显然，PLR 法只能用于比较两个预计模型的相对预计性

PLR= 3.31401829985515E-167

$i \to \infty$，PLR的值趋近于 0，所以 G-O 模型更优。

4.2 U 图法

U 图示用来检测预计和观测的失效行为之间系统而客观的差别的，U 图的目的是用来判断预测预计分布函数$\hat{F_j}(t)$是否均匀地接近于实际分布$F_j(t)$，U 图就是序列$\{u_j\}$的抽样分布函数图，用来表示$\{u_j\}$的抽样累积分布函数接近 U(0,1)的累积分布函数的程度。

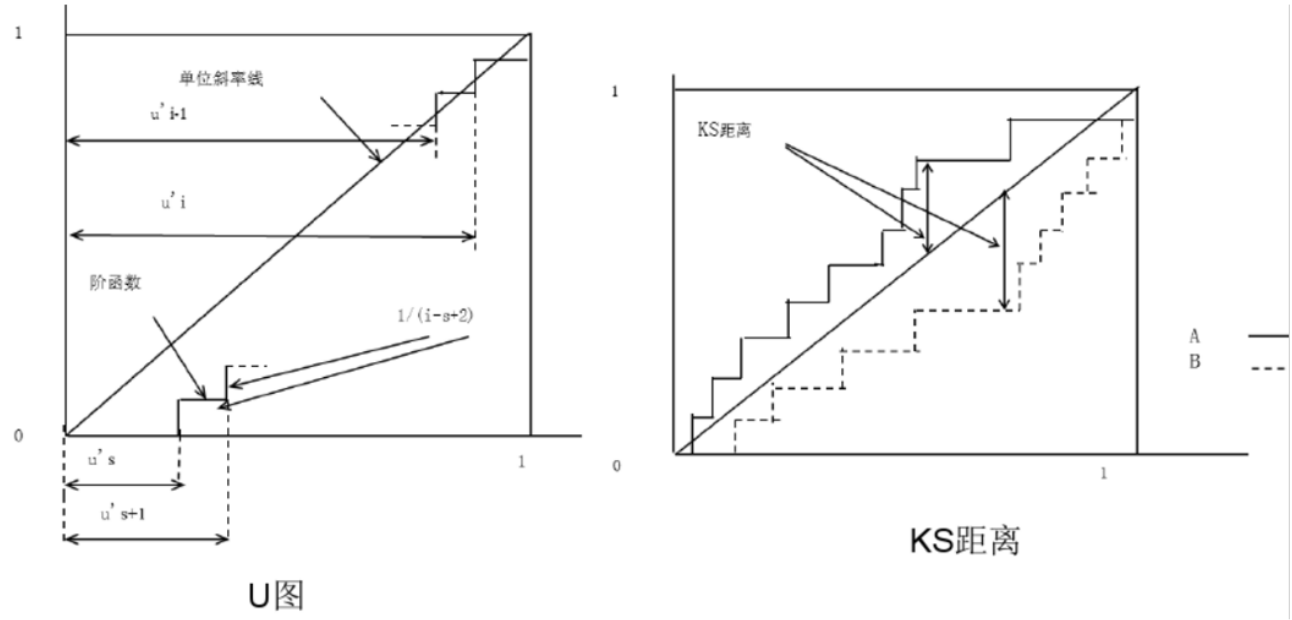

图 2. u-结构图、KS 距离图

J-M 模型的预计分布函数为：$F(x_i) = 1 - exp\{-\psi(N_0 - i + 1)x_i\}$
G-O 模型的预计分布函数为：$F(t) = 1 - e^{a(e^{-b(t+s)} - e^{-bt})}$

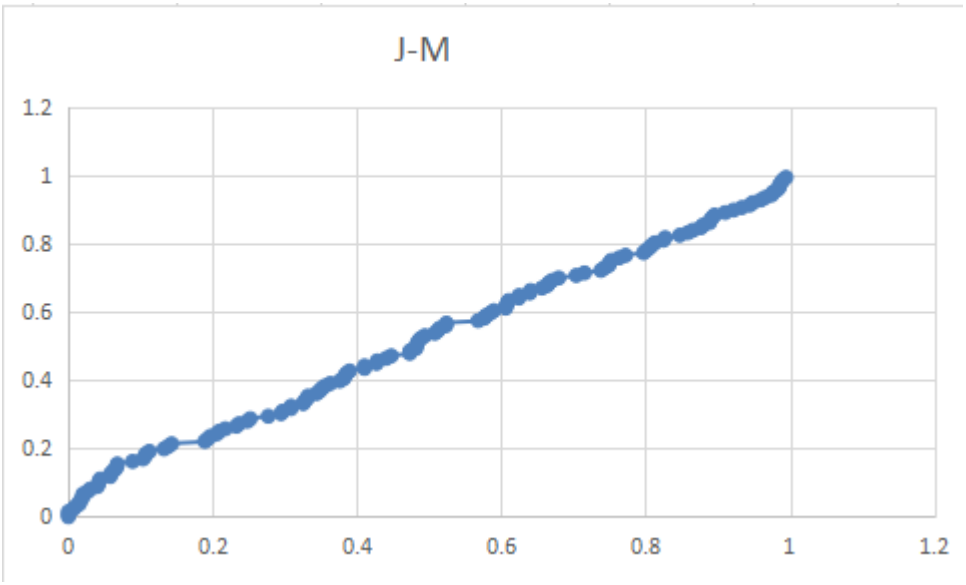

图 3. JM-u 结构图

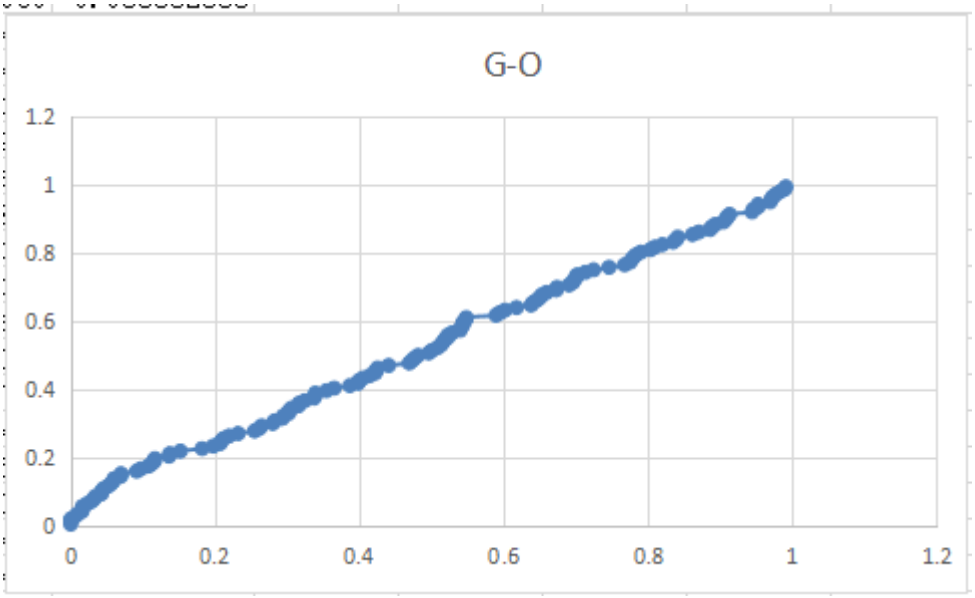

图 4. GO-u 结构图

求得 KS 距离

JM 的图

KS = 0.9051761554045642

GO 的图

KS = 0.9060508810007392

Y 图

U 图能够检测出预计与显示的严重偏差，但 KS 距离很小的 U 图有时也会掩盖了偏差，需要求助于 Y 图。Y 图示通过序列u_j进行变换后绘制的。

Y 图对$\{u_j\}$作如下的进一步变换

$$x_j = -ln(1-u_j) \qquad y_k = \frac{\sum_{j=s}^{k} x_j}{\sum_{j=s}^{i} x_j}$$

Y-图表示$\{u_j\}$的样本累积分布函数与 45 度线的接近程度

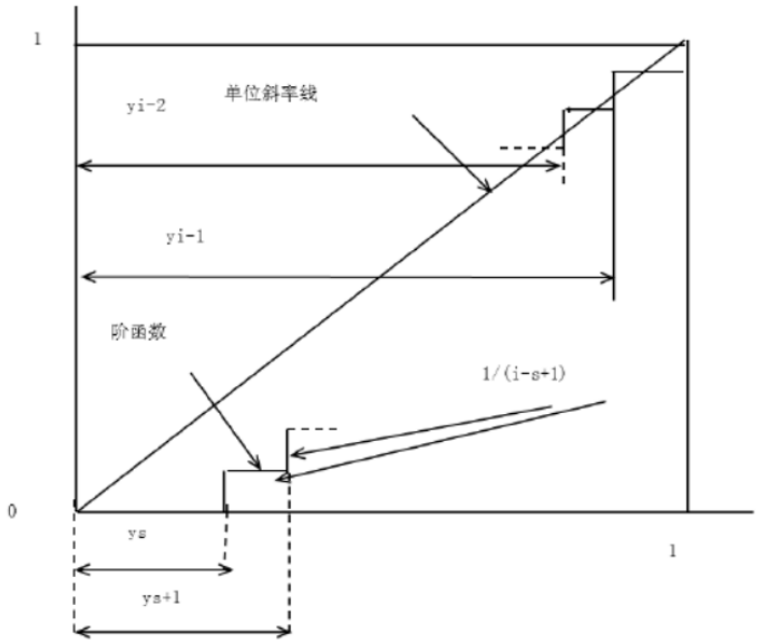

图 5. Y 图示意图

拟合曲线可得

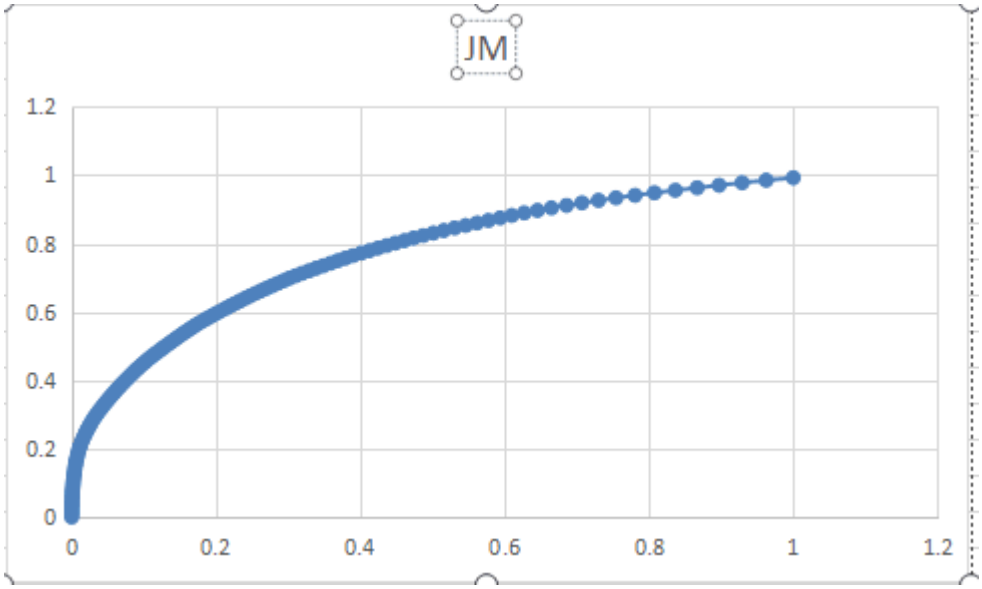

图 6. JM-Y 结构图

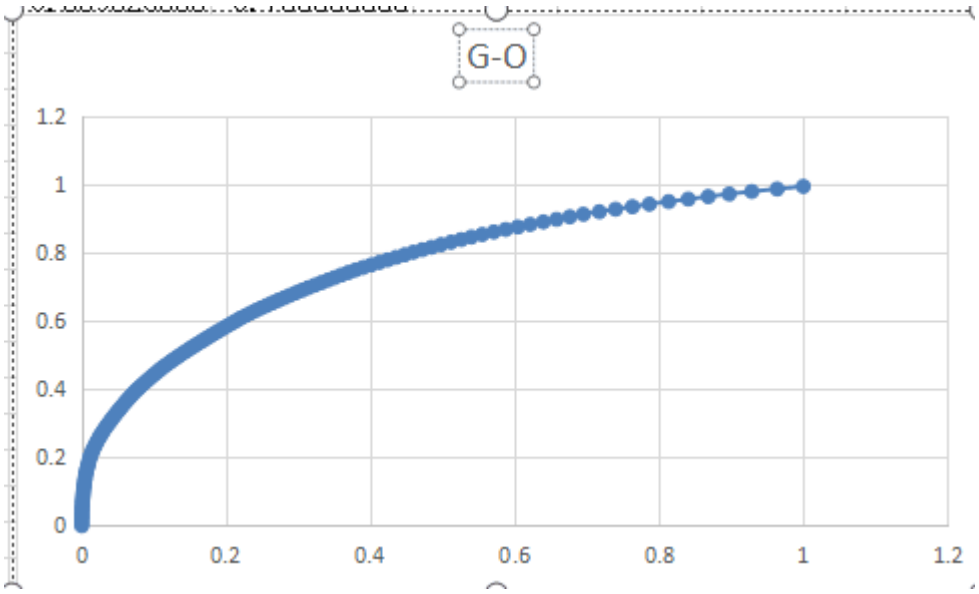

图 7. JM-Y 结构图

求得 KS 距离

JM 的y图

KS = 0.39968676358616834

GO 的y图

KS = 0.3884802491649051

可看出 GO 更加好一点

参 考 文 献

[1] (中)张德平编著;软件系统可靠性分析基础与实践（Fundamentals and Practice of Software System Reliablity Analysis）清华大学出版社

[2]（中）宋晓秋.软件可靠性 GO 模型的特性分析[J].计算机工程与设计, 1996, 17(6):3.DOI:CNKI:SUN:SJSJ.0.1996-06-008.

[3]谢尚飞,孙志礼,杨丽.经典软件可靠性模型分析[J].软件工程师, 2012(12):3.DOI:10.3969/j.issn.1008-0775.2012.12.008.