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： J-M 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 17 日

目录

1.引言 3
1.1 编写目的 3
2.模型理论 3
2.1 模型背景 3
2.2 模型假设 4
2.3 模型推导 4
2.4 算法原理 4
2.5 算法步骤 5
3.算法实现 5
3.1 算法流程图 5
3.2 算法伪代码 7
3.3 算法实现 8
3.5 数据集 15
3.6 算法结果分析 15
4.总结 17
参 考 文 献 17

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 JM 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 JM 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

该模型由 Jelinski 和 Moranda 于 1972 年开发，是最早提出的软件可靠性模型之一，具有重要的历史意义。它最初应用于麦克唐奈·道格拉斯（McDonnell Douglas）海军工程项目中，标志着软件可靠性评估领域的一次重要突破。这一模型的提出，为后来的软件可靠性研究和工程实践奠定了理论基础，推动了该领域的深入发展。

该模型以一种简明而直观的方式展示了如何通过软件缺陷的显露历程，来预测软件在未来运行中的可靠性表现。它从实际出发，提出了许多关于软件缺陷演化过程的假设，并基于这些假设构建了一个可操作的可靠性预测框架。这些假设成为了软件可靠性建模的核心基础，因此该模型也催生了众多变体和扩展版本。

实际上，现有的多数软件可靠性模型，或多或少都可以看作是对 Jelinski-Moranda 模型的变形或扩展，或与之有着密切的关系。例如，许多后续的模型在此基础上增加了新的因素，或者在假设条件上做了调整，以适应不同的软件系统和应用场景。因此，Jelinski-Moranda 模型不仅为后续的理论发展提供了方向，也为实践中解决软件可靠性问题提供了指导。

总的来说，Jelinski-Moranda 模型在软件可靠性定量分析技术的建立与发展中，做出了不可或缺的贡献。它的提出标志着软件可靠性研究领域的第一个里程碑，为整个行业提供了重要的理论支持和方法论基础，影响深远。

2.2 模型假设

1) 程序中的固有的初始错误个数 N_0 为一个未知的常数;
2) 程序中的各个错误是相互独立的，每个错误导致系统发生失效的可能性大致相同，各次失效间隔时间也相互独立;
3) 测试过程中错误一旦被检测出即被完全排除，每次排错只排除一个错误，排错时间可以忽略不计，在排错过程中不引入新的错误
4) 程序的失效率在每个失效间隔时间内是常数，其数值正比于程序中残留的错误数，比例系数为Φ;
5) 软件的运行方式与预期的运行方式相似

2.3 模型推导

JM 模型给出的失效率函数为：

$$Z(x_i) = \Phi(N_0 - i + 1) \tag{1}$$

其中Φ和 N_0 是未知参数，也就是我们要进行估计的参数，以第 i-1 次失效为起点的第 i 次失效发生的时间X_i是一个随机变量，它服从以$Z(x_i)$为参数的指数分布，其概率密度函数为：

$$f(x_i) = \Phi(N_0 - i + 1)exp\{-\Phi(N_0 - i + 1)x_i\} \tag{2}$$

用最大似然法对上述两个未知参数进行估计，最终得到的估计值是从下列两个方程中求解的：

$$\begin{cases} \widehat{\Phi} = \dfrac{n}{\widehat{N}(\sum_{i=1}^{n} x_i) - \sum_{i=1}^{n}(i-1)x_i} \\ \sum_{i=1}^{n} \dfrac{1}{\widehat{N} - (i-1)} = \dfrac{n}{\widehat{N} - \left(\dfrac{1}{\sum_{i=1}^{n} x_i}\right)(\sum_{i=1}^{n}(i-1)x_i)} \end{cases} \tag{3}$$

2.4 算法原理

由于上述方程难以直接解析求解，需采用数值方法迭代计算N_0和Φ。定义辅助函数$f(N)$

$$f(N) = \sum_{i=1}^{n} \frac{1}{N - i + 1} - \frac{n}{N - p} \tag{4}$$

其中

$$p = \frac{\sum_{i=1}^{n}(i-1)(t_i - t_{i-1})}{t_n} \tag{5}$$

通过数值算法迭代逼近使$f(N)$收敛于零，得到N_0。

2.5 算法步骤

1) 步骤 1：如果$P > \frac{n-1}{2}$, 则$left = n-1, right = n$, 转步骤 2；如果$P \le \frac{n-1}{2}$, 则终止计算。

2) 步骤 2：如果$f(right) > e_y$, 则$left = right$且$right = right + 1$, 重复步骤 2；若$-e_x \le f(right) \le e_y$, 则$root = right$,转步骤 4；若$f(right) \le -e_x$, 转步骤 3。

3) 步骤 3:如果$|right - left| < e_x$, 则$root = \frac{right+left}{2}$, 转步骤 5;如果$|right - left| > e_x$, 则$root = \frac{root+left}{2}$, 转步骤 4。

4) 步骤 4:如果$f(root) > e_y$,则$left = root$，转步骤 3；如果$-e_y \le f(root) \le e_y$,则转步骤 5；如果$f(root) < -e_y$,则$right = root$,转步骤 3。

5) 步骤 5:$N = root, \Phi = \frac{n}{N_n - \sum_{i=1}^{n}(i-1)(t_i - t_{i-1})}$,终止计算。

其中，e_x为自变量x的误差精度，e_y为自变量y的误差精度。

3.算法实现

3.1 算法流程图

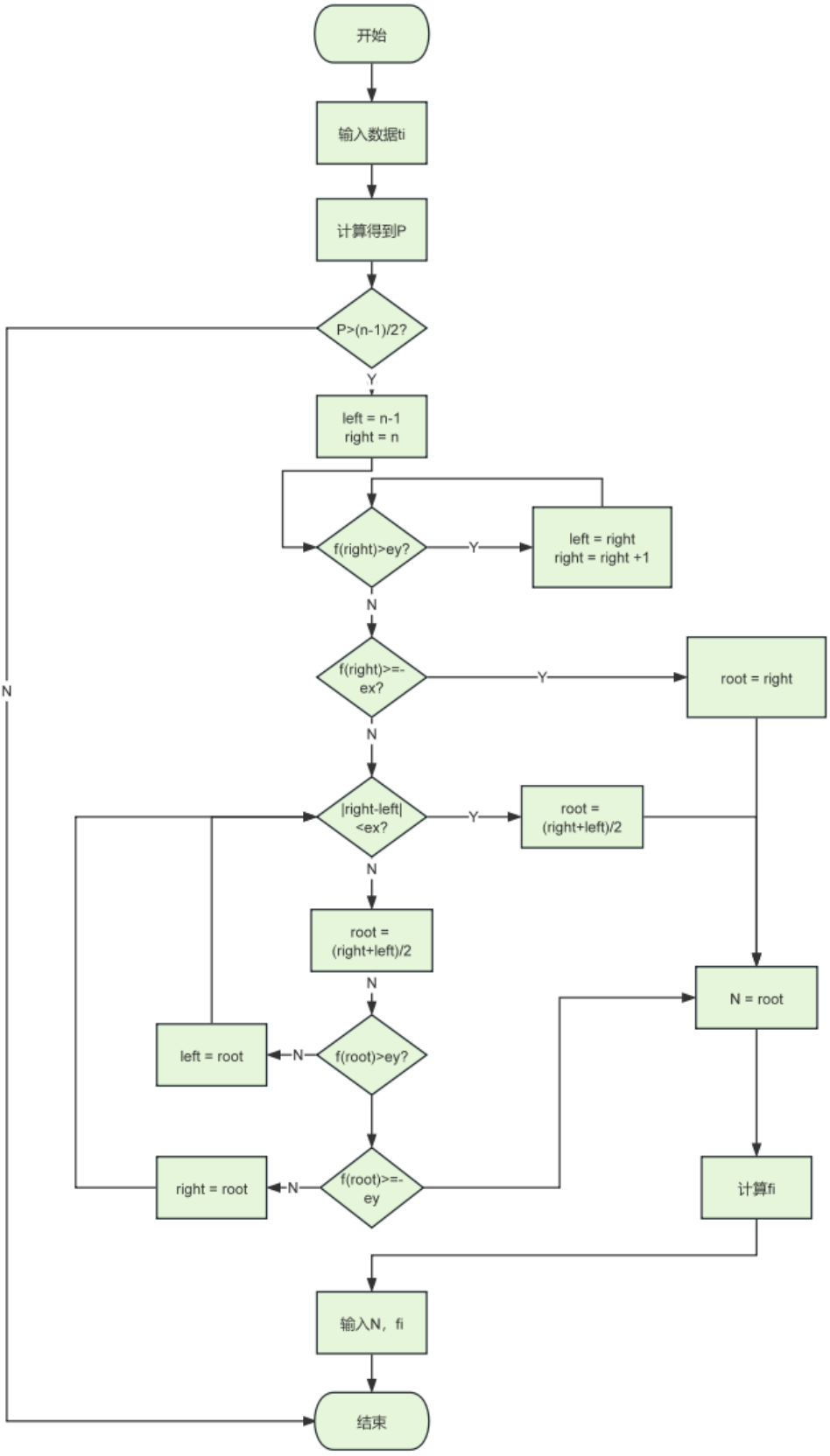

图 1.JM 模型算法流程图

3.2 算法伪代码

$$f(N)=\sum_{i=1}^{n}\frac{1}{N-i+1}-\frac{n}{N-p}\quad p=\frac{\sum_{i=1}^{n}(i-1)(t_i-t_{i-1})}{t_n} \tag{6}$$

首先需要计算 P 值，这在函数 getP()中实现，

```
    // 计算P的值
    public double getP(JM jm) {
        double result = 0;
        double sum = 0;
        int n = jm.T.size() - 1;
        for (int i = 1; i <= n; i++) {
            sum += (i - 1) * (jm.T.get(i) - jm.T.get(i - 1));
        }
        result = sum / jm.T.get(n);
        return result;
    }
```

还需要一个 f()函数，实现上式

```
    // 求函数f（N）
    public double fun(JM jm, double k, double P) {
        double result = 0;
        double sum = 0;
        int n = jm.T.size() - 1;
        for (int i = 1; i <= n; i++) {
            sum += (1 / (k - i + 1));
        }
        result = sum - n / (k - P);
        return result;
    }
```

本程序最重要的就是五个步骤的执行，其伪代码如下：

```
//步骤1
if (P <= ((n - 1) / 2))
return
//步骤2
while (f (t, right) > ey) {
left = right;
    right = right + 1;
}
if (f (t, right) >= (-ex))
root = right;
else{
while (1) {
//步骤3
        if (|right - left| < ex) {
```

```
            root = (right + left) / 2;
            break;
        }
        root = (right + left) / 2;
//步骤4
        if (f (t, root) > ey)
            left = root;
        else {
            if (f (t, root) < (-ey))
                right = root;
            else break;
        }
    }
}
//步骤5
N = root;//得到N
tn = t.get(n - 1);//tn 是输入数据
for ( i = 1; i < n; i++) {
    t1 = t.get(i);
    t2 = t.get(i - 1);
    sum = sum + (i - 1) * (t1 - t2);
}
Φ= (n - 1) / (N * tn - sum);//得到Φ
```

3.3 算法实现

本次实验采用 Java 语言编写，运行的 IDE 是 vscode。

算法关键代码如下：

```
    // 计算P的值
    public double getP(JM jm) {
        double result = 0;
        double sum = 0;
        int n = jm.T.size() - 1;
        for (int i = 1; i <= n; i++) {
            sum += (i - 1) * (jm.T.get(i) - jm.T.get(i - 1));
        }
        result = sum / jm.T.get(n);
        return result;
    }

    // 求函数f（N）
    public double fun(JM jm, double k, double P) {
```

```
        double result = 0;
        double sum = 0;
        int n = jm.T.size() - 1;
        for (int i = 1; i <= n; i++) {
            sum += (1 / (k - i + 1));
        }
        result = sum - n / (k - P);
        return result;
    }
    // 计算N的值
    public void setN(JM jm) {
        int n = jm.T.size() - 1;
        double left = 0;
        double right = 0;
        double root = 0;
        double f_left = 0;
        double f_right = 0;
        double f_root = 0;
        double P;
        P = jm.getP(jm);
        System.out.println(P);
        // 步骤1
        if (P > (n - 1) / 2) {
            left = n - 1;
            right = n;
        } else
            return;
        // 步骤2
        f_left = jm.fun(jm, left, P);
        f_right = jm.fun(jm, right, P);
        while (f_right > jm.ey) {
            left = right;
            right = right + 1;
            f_right = jm.fun(jm, right, P);
        }
        if (-jm.ex <= f_right) {
            root = right;
            jm.N = root;
            return;
        } else {
            while (true) {
                // 步骤3
                if (Math.abs(right - left) < jm.ex) {
                    root = (right + left) / 2;
```

```
                    break;
                }
                if (Math.abs(right - left) > jm.ex) {
                    root = (right + left) / 2;
                }
                // 步骤4
                f_root = jm.fun(jm, root, P);
                if (f_root > jm.ey) {
                    left = root;
                    continue;
                }
                if (-jm.ey <= f_root && f_root <= jm.ey) {
                    jm.N = root;
                    break;
                }
                if (f_root < -jm.ey) {
                    right = root;
                    continue;
                }
            }
        }
        jm.N = root;
        return;
    }
    // 计算Φ的值
    public void setFi(JM jm) {
        double sum = 0;
        int n = jm.T.size() - 1;
        for (int i = 1; i <= n; i++) {
            sum += (i - 1) * (jm.T.get(i) - jm.T.get(i - 1));
        }
        jm.Fi = n / (jm.N * jm.T.get(n) - sum);
    }
    public void setF(JM jm) {
        int n = jm.T.size();
        int k = 10;
        double index = 0;
        double result = 0;
        int j = n - 1;
        for (int i = 0; i < k; i++) {
            index = -jm.Fi * (jm.N - j) * jm.T.get(j);
            j--;
            result = 1 - Math.exp(index);
            jm.F.add(i, result);
```

```
        }
        System.out.println(jm.F.size());
        for (int i = 0; i < k; i++) {
            System.out.println(jm.F.get(i));
        }
        return;
    }
```

完整代码如下：

```
package softwarereality;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
public class JM {
    // 使用List存储数据T[i]
    ArrayList<Double> T = new ArrayList<Double>();
    ArrayList<Double> F = new ArrayList<Double>();
    double N = 0;
    double Fi = 0;
    double ex = 0.1;
    double ey = 0.1;
    // 从文本文件中读取数据
    public void setT(JM jm) {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader("D:\\javacode
\\softwarereality\\jm.txt"));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        String line = null;
        try {
            line = br.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
        double temp = 0;
        jm.T.add(0.0);
        while (line != null) {
            String[] numbers = line.split("\\s+");
            temp += Double.valueOf(numbers[1]);
            jm.T.add(temp);
            try {
```

```
                line = br.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        try {
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println(jm.T.size());
    }
    // 计算P的值
    public double getP(JM jm) {
        double result = 0;
        double sum = 0;
        int n = jm.T.size() - 1;
        for (int i = 1; i <= n; i++) {
            sum += (i - 1) * (jm.T.get(i) - jm.T.get(i - 1));
        }
        result = sum / jm.T.get(n);
        return result;
    }
    // 求函数f（N）
    public double fun(JM jm, double k, double P) {
        double result = 0;
        double sum = 0;
        int n = jm.T.size() - 1;
        for (int i = 1; i <= n; i++) {
            sum += (1 / (k - i + 1));
        }
        result = sum - n / (k - P);
        return result;
    }
    // 计算N的值
    public void setN(JM jm) {
        int n = jm.T.size() - 1;
        double left = 0;
        double right = 0;
        double root = 0;
        double f_left = 0;
        double f_right = 0;
        double f_root = 0;
        double P;
```

```
        P = jm.getP(jm);
        System.out.println(P);
        // 步骤1
        if (P > (n - 1) / 2) {
            left = n - 1;
            right = n;
        } else
            return;
        // 步骤2
        f_left = jm.fun(jm, left, P);
        f_right = jm.fun(jm, right, P);
        while (f_right > jm.ey) {
            left = right;
            right = right + 1;
            f_right = jm.fun(jm, right, P);
        }
        if (-jm.ex <= f_right) {
            root = right;
            jm.N = root;
            return;
        } else {
            while (true) {
                // 步骤3
                if (Math.abs(right - left) < jm.ex) {
                    root = (right + left) / 2;
                    break;
                }
                if (Math.abs(right - left) > jm.ex) {
                    root = (right + left) / 2;
                }
                // 步骤4
                f_root = jm.fun(jm, root, P);
                if (f_root > jm.ey) {
                    left = root;
                    continue;
                }
                if (-jm.ey <= f_root && f_root <= jm.ey) {
                    jm.N = root;
                    break;
                }
                if (f_root < -jm.ey) {
                    right = root;
                    continue;
                }
```

```
            }
        }
        jm.N = root;
        return;
    }
    // 计算ϕ的值
    public void setFi(JM jm) {
        double sum = 0;
        int n = jm.T.size() - 1;
        for (int i = 1; i <= n; i++) {
            sum += (i - 1) * (jm.T.get(i) - jm.T.get(i - 1));
        }
        jm.Fi = n / (jm.N * jm.T.get(n) - sum);
    }
    public void setF(JM jm) {
        int n = jm.T.size();
        int k = 10;
        double index = 0;
        double result = 0;
        int j = n - 1;
        for (int i = 0; i < k; i++) {
            index = -jm.Fi * (jm.N - j) * jm.T.get(j);
            j--;
            result = 1 - Math.exp(index);
            jm.F.add(i, result);
        }
        System.out.println(jm.F.size());
        for (int i = 0; i < k; i++) {
            System.out.println(jm.F.get(i));
        }
        return;
    }
    public static void main(String[] args) {
        JM jm = new JM();
        jm.setT(jm);
        jm.setN(jm);
        jm.setFi(jm);
        // jm.setF(jm);
        System.out.println("while ex = " + jm.ex + ",ey = " + jm.ey);
        System.out.println("N = " + jm.N);
        System.out.println("ϕ = " + jm.Fi);
    }
}
```

3.5 数据集

表 1. NTDS 数据集

错误数	错误间隔时间	错误数	错误间隔时间	错误数	错误间隔时间
1	9	13	1	25	2
2	12	14	9	26	1
3	11	15	4	27	87
4	4	16	1	28	47
5	7	17	3	29	12
6	2	18	3	30	9
7	5	19	6	31	135
8	8	20	1	32	258
9	5	21	1	33	16
10	7	22	33	34	35
11	1	23	7		
12	6	24	91		

这里以美军海军战术数据系统（Naval Tactical Data System，NTDS）数据集为例，其数据集形式如表 1 所示。

3.6 算法结果分析

```
while ex = 0.001,ey = 0.001
N = 34.0029296875
ϕ = 0.004845039046567121
```

验证：将上述结果代入方程式

```
Received Output:
4.11344
4.11349
0.00484504
```

CPP 验证程序如下

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
#define endl '\n'
const int mod = 1e9 + 7;
const int N = 2e5 + 10;

void solve()
{
    int n;
```

```
    cin >> n;
    vector<double> x(n + 1);
    for (int i = 1; i <= n; ++i)
        cin >> x[i];
    double sumx = 0;
    double sumx1 = 0;
    // 等式右侧
    for (int i = 1; i <= n; ++i)
        sumx += x[i];
    for (int i = 1; i <= n; ++i)
        sumx1 += (double)(i - 1) * x[i];
    // 等式左侧
    double sumx2 = 0;
    for (int i = 1; i <= n; ++i)
        sumx2 += 1.0 / (34.0029296875 - (i - 1));
    cout << 1.0 * n / (34.0029296875 - (1.0 / sumx) * (sumx1)) << endl;
    cout << sumx2 << endl;
    cout << 1.0 * n / (34.0029296875 * sumx - sumx1) << endl;
}
int main(void)
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int t = 1;
    // cin >> t;
    while (t--)
        solve();
    return 0;
}
```

在误差范围内，算法正确。

其它精度范围结果：

```
while ex = 0.01,ey = 0.01
N = 34.0
ϕ = 0.004846756949394155
```

```
while ex = 0.1,ey = 0.1
N = 34.0
ϕ = 0.004846756949394155
```

```
while ex = 1.0E-4,ey = 1.0E-4
N = 34.0029296875
ϕ = 0.004845039046567121
```

4.总结

我们可以估计该程序中的固有错误N_0=34;失效率函数的比例常数Φ为 0.004845 左右；失效间隔时间的估计值可由以下式子算出：

$$\boldsymbol{MTBF = 1/(0.004845)(34 - i + 1);}$$

通过改变e_x和e_y的值，(即针对自变量 x,y 的误差精度控制值)，输出结果有一定的差异。由实验结果我们可以看出，ex 和 ey 对结果是有一定影响的，但影响随着 ex、ey 的增长越来越小，精度越精，误差越小。

由此看来 JM 模型在评估开发结束时软件的可靠性水平 MTBF 或失效率还是十分可靠的。

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J—M增长模型的特性分析[J].系统工程与电子技术, 1997, 19(9):4.