南京航空航天大学
Nanjing University of Aeronautics and Astronautics

软件可靠性课程实验报告

题　　目:　G-O 软件可靠性模型

院　　系:　计算机科学与技术学院

专　　业:　软件工程

学生姓名:　XX

学　　号:　1617303XX

二零一九 年 十月 十日

目录

1.引言

1．1编写目的

随着软件规模越来越大，结构日趋复杂，应用日趋广泛。软件危机依然是我们难以逾越的障碍， 加强软件工程管理，势在必行，势在必然！改进和提高软件可靠性，为部队提供可靠顶用的装备是我们的义务和责任！

此次试验以GO模型为核心，帮助我们理解什么是软件可靠性评估及其基本原理，掌握和应用GO模型。

2.模型理论

2.1 模型背景

累积故障数 N(t),作为时间的变化量, 可以用一个 NHP P 过程模型加以描述。NHPP模型在许多实际的应用中, 特别是硬件可靠性分析中, 为许多学者及工程人员所采用。有不少软件可靠性模型也属于这一类。利用 NHPP模型, 可以在一定前提条件下, 对许多软件可靠性模型中的参数予以推导。

最著名的 NHPP模型应算 G-O 模型,它于1979 年由 Goel 和 Okumot 提出, 之后又有许多人提出了不少类似的模型。下面我们将详细介绍 G-O 模型, 并以它为基础, 介绍利用 NHPP 模型怎样进行估测及参数估计的方法,之后我们再对其它 NHPP 模型进行概述。

A. L. Goel 和 K. Okumoto 于 1979 年提出关于连续时间的非齐次泊松过程(Nonhomogeneous Poisson Process) 模型,简称为 NHPP类的 G-O 模型。他们的工作对于随后的软件可靠性模型的建立有着很大的影响, NHPP类模型已成为软件可靠性模型的一个大类。

2.2 模型假设

- 测试未运行的软件失效为0；当测试进行时，软件失效服从均值为m(t)的泊松分布；
- 当△t 趋于0时，测试时间（t,t+△t）内产生的失效与软件残留错误成正比；
- 对于任一组有限时间点,在对应时间段分别产发生的失效次数相互独立；

- 每次只修正一个错误，当软件故障出现时，引发故障的错误立刻被排除，并不会引入新的错误；

2.3 模型推导

GO 模型在测试区[0,t]内的累计失效数期望函数为：$m(t)=a(1-\exp(-bt))$，t 为软件累计测量时间；可靠性函数为 R（$s\mid$t）$=e^{a(e^{-b(t+s)}-e^{-bt})}=e^{-m(s)e^{-bt}}$

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

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

则 t 的联合概率密度函数为：

$$f_{t1,t2,\ldots\ldots,\ tm}(t_{1,}t_{2,}\ldots\ldots t_{m})=\prod_{i=1}^{m}abe^{-bt_{i}}\exp[-a(1-e^{-bt_{m}})]$$

则在给定 t 时，关于参数 a,b 的似然函数为：

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

得到以下方程：

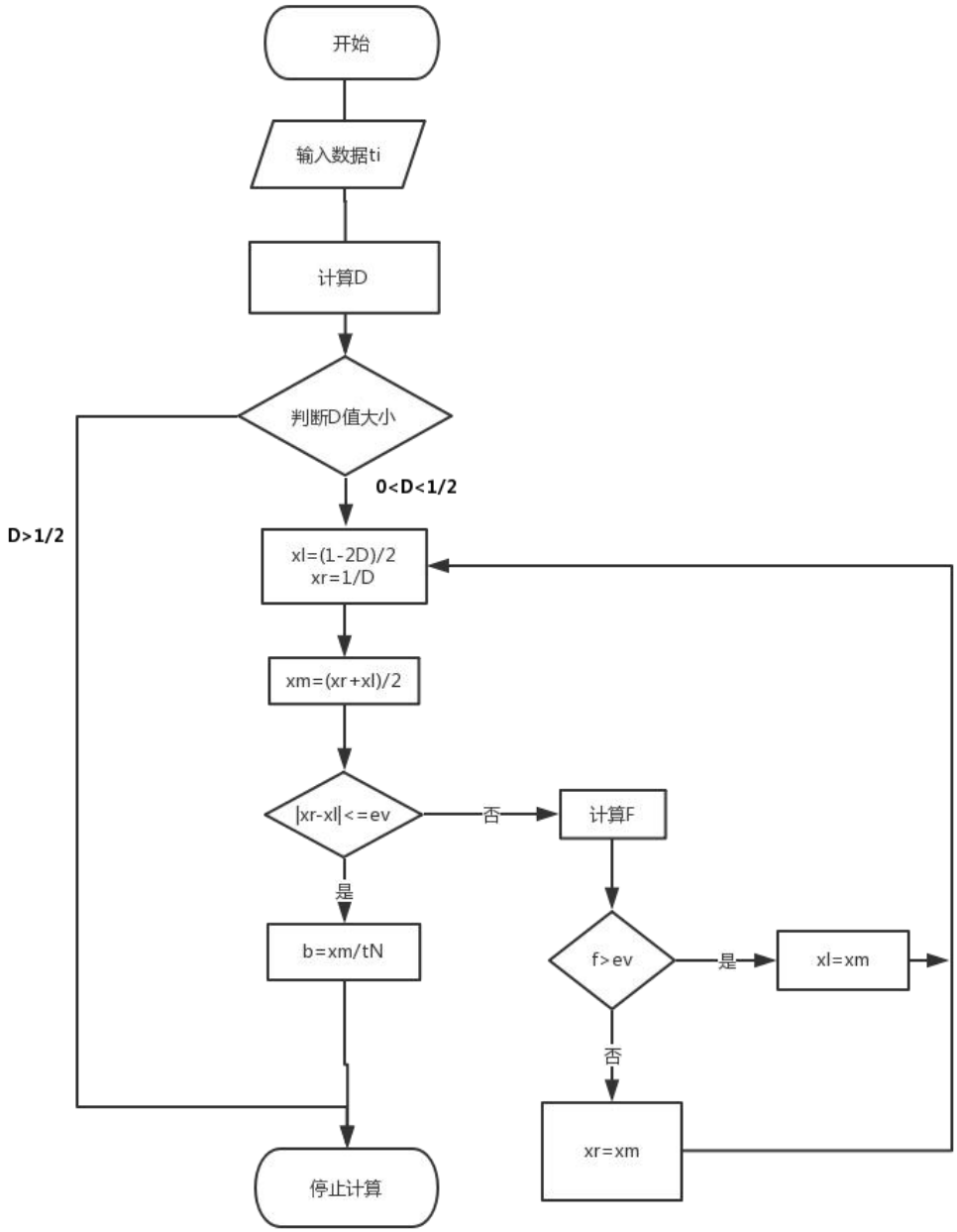
$\frac{N}{\mathrm{a}}=1-e^{-bt_{m}}$，

$\frac{N}{b}=\sum_{\mathrm{i}=1}^{m}t_{i}+at_{m}e^{-bt_{m}}$

所以该方程的解就是参数 a 和 b 的估计值；

3.算法实现

3.1 流程图


开始
输入数据ti
计算D
判断D值大小
D>1/2
0<D<1/2
xl=(1-2D)/2
xr=1/D
xm=(xr+xl)/2
|xr-xl|<=ev
否
计算F
是
b=xm/tN
f>ev
是
xl=xm
否
xr=xm
停止计算

3.2 算法伪代码

$$D=\sum_{i=0}^{N} \mathrm{t}_{\mathrm{i}} / Nt_N$$

首先需要计算 D 值，这在函数 Dfunction()中实现，

```
sum=0;
for(i=1 to n){
sum=sum+ti;
}
sum=sum/(N*tn);
```

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
求 D
if (D >=1/2))
    Return
else if(0<D<1/2)
{Xl=(1-2D)/2;
Xr=1/D;
}

//步骤 2
While(true)
{Xm=(xr+xl)/2
//步骤 3
If(|xr - xl| >= ev)
{计算 f;
If(f>ev)
{xl=xm;
Break;
}
Else if(f<-ev)
{
xr=xm;
break;
}
}

//步骤 4
```

```
  if (|xr - xl| < ev) {
            b=xm/tN;a=N/(1-exp(-btN));
            break;
        }


}
//步骤 5
输出 a,b;停止计算
```

3.3 算法实现

本次实验采用 Java 语言编写，运行的 IDE 是 IDEA。

数据结构：

```
//用double类型的ArrayList来存储ti
    ArrayList<Double> t=new ArrayList<Double>();
```

数据的初始化：

```
int N=t.size();
 double tn=t.get(N-1);
 double xl=0;
 double xr=0;
 double xm=0;
 double epslv=0.1;
 double a=0;
 double b=0;
 double D=Dfunction(t);
 double f=0;
```

实现的关键代码如图所示，首先是主函数中代码：

```
        double f=0;
//步骤1:判断D大小
        if(D>0&&D<0.5)
        {
            xl=(1-2*D)/2;
            xr=1/D;

        }
        else {

            input.close();//终止计算
            return ;}
//转步骤2:
```

```
            return ;}
//转步骤2:

        while (true)
        {       xm=(xl+xr)/2;
            if(Math.abs(xr-xl)<=epslv)
                break;
            else{
                double y=Math.exp(xm);
                //步骤3
                f=(1-D*xm)*y+(D-1)*xm-1;
                if(f>epslv)
                {
                    xl=xm;
                    continue;
                }
                else if(f<-epslv)
                {
                    xr=xm;
                    continue;
                }

            }

        }
```

```
        }
//步骤4
        b=xm/tn;
        double ahelp=Math.exp(-b*tn);
        a=N/(1-ahelp);
        System.out.println("a="+a);
        System.out.println("b="+b);

input.close();
    }
```

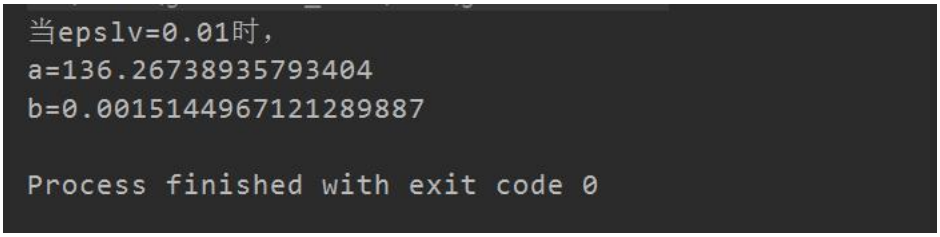
Dfunction 函数中代码如下：

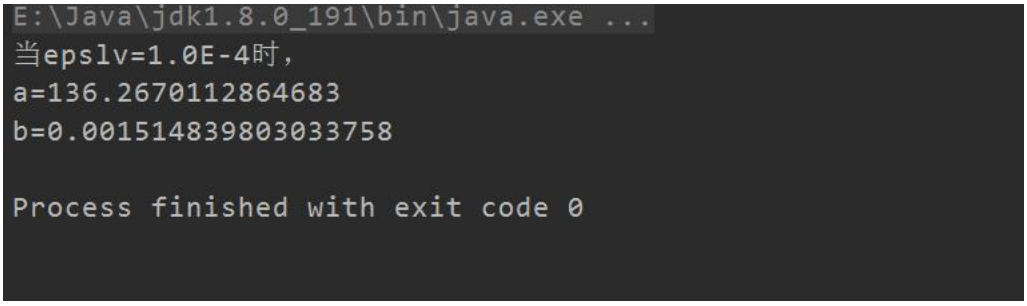
```
public static double Dfunction(ArrayList<Double> list)
{
    double result=0;
    int n=list.size();
    for(int i=1;i<n;i++)
    {double t1=list.get(i);
      result+=t1;

    }
    double tn=list.get(n-1);
    result=result/tn;
    result=result/n;
    return result;


}
```

3.5 数据来源

数据来源是老师提供的失效错误数据 SYS1(failue_count).txt，我已放入了 java 工程目录下。

3.6 结果展示





4.总结

我们可以估计累计失效数期望函数的比例常数 a 为 136 左右,b 为 0.0015；

通过改变 ε 的值，输出结果有一定的差异。由实验结果我们可以看出，

ε 对结果是有一定影响的，但影响随着 ε 的增长越来越小，精度越精，误差越小。

由此看来 GO 模型在评估开发时还是十分可靠的。

参考文献

[1] (美)[M.R.柳]Michael R.Lyu 主编; 刘喜成,钟婉懿等译.软件可靠性工程手册（Handbook Of Software Reliabilty Engineering）[M].电子工业出版社，1997