

**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:** 徐宇顺

**学 号:** 031720106

**二零一九 年 十 月 六日**

目录

[1.引言 3](#_Toc529726265)

[1．1编写目的 3](#_Toc529726266)

[2.模型理论 3](#_Toc529726267)

[2.1 模型背景 3](#_Toc529726268)

[2.2 模型假设 3](#_Toc529726269)

[2.3 模型推导 4](#_Toc529726270)

[3.算法实现 5](#_Toc529726271)

[3.1 流程图 5](#_Toc529726272)

[3.2 算法伪代码 6](#_Toc529726273)

[3.3 算法实现 7](#_Toc529726274)

[3.5 数据来源 8](#_Toc529726275)

[3.6 结果展示 8](#_Toc529726276)

[4.总结 9](#_Toc529726277)

[参考文献 9](#_Toc529726278)

# 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！

# 2.模型理论

## 2.1 模型背景

Goel-Okumto软件可靠性模型（G-O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

* 测试未运行的软件生效为0；当测试进行时，软件失效服从均值为m（t）的泊松分布
* 当δt ->0时，测试时间（t,t+δt）内产生的失效与软件残留错误成对比
* 对于任一组有限时间点，在对应时间段分别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，引发故障的错误被立即排除，并不会引入新的错误

## 2.3 模型推导

G\_O模型在测试区间[0,t]内的累计失效数期望函数为



t为软件累计测试时间。

可靠性函数为：



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:



得出参数的似然函数为:



步骤1：令给定的误差允许值为，如果0<D<1/2，则，故，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，如果，则转步骤4

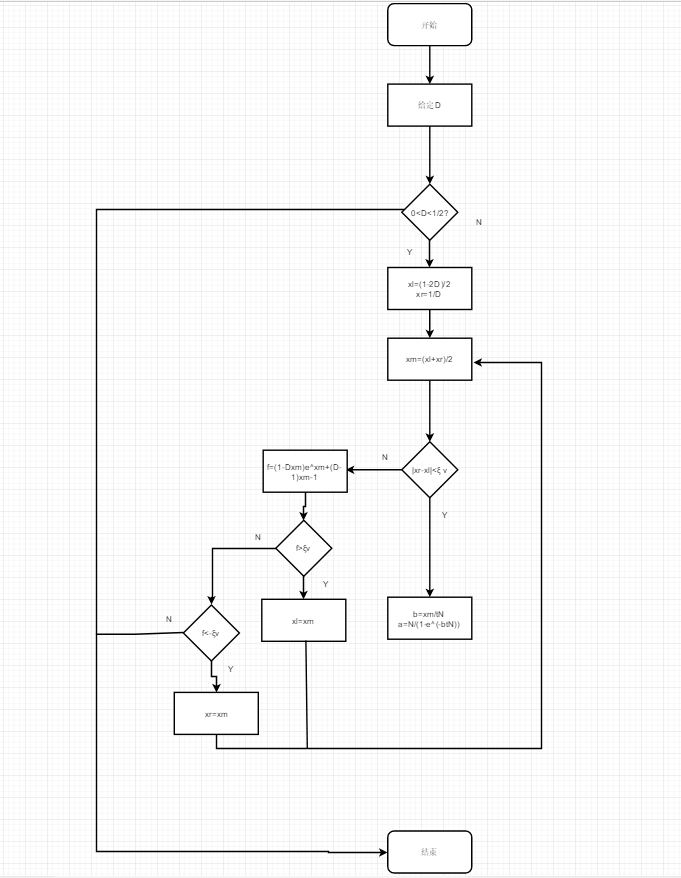
步骤3：，如果，则转步骤2；如果，则，转步骤2

步骤4：计算和

步骤5：停止计算

# 3.算法实现

## 3.1 流程图

****

## 3.2 算法伪代码

## 







## 3.3 算法实现

本次实验采用Java语言编写，运行的IDE是IDEA。

**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>();  
 **double D**=0;  
 **double f**=0;  
 **double xl**=0;  
 **double xr**=0;  
 **double xm**=0;  
 **double b**=0;  
 **double a**=0;  
 **double v**=0.1;  
  
  
 **public void** setT(G\_O g\_o) {  
 BufferedReader br = **null**;  
 **try** {  
 br = **new** BufferedReader(**new** FileReader(**"D:\\小黄片\\软件可靠性\\实验\\2\\src\\test.txt"**));  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 **double** temp=0;  
 String line = **null**;  
 **try** {  
 line = br.readLine();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 String[] T0 = line.split(**"\\s+"**);  
 temp=Double.valueOf(T0[1]);  
 g\_o.T.add(temp);  
 **try** {  
 line = br.readLine();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 **while**(line != **null**){  
 String[] numbers = line.split(**"\\s+"**);  
 temp+=Double.valueOf(numbers[1]);  
 g\_o.T.add(temp);  
 **try** {  
 line = br.readLine();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **try** {  
 br.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
}

**public double** getF(G\_O g\_o,**double** D){  
 **double** p1,p2;  
 **double** result;  
 p1=(1-D\*g\_o.**xm**)\*(Math.*pow*(Math.***E***,g\_o.**xm**));  
 p2=(D-1)\*g\_o.**xm**-1;  
 result=p1+p2;  
 **return** result;  
 }  
  
 **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;  
 }  
  
 **public double** geta(G\_O g\_o){  
 **double** result,temp1,temp2;  
 **int** n=g\_o.**T**.size();  
 temp2=-**b**\*(g\_o.**T**.get(n-1));  
 temp1=Math.*pow*(Math.***E***,temp2);  
 result=n/(1-temp1);  
 **return** result;  
 }  
  
 **public static void** main(String[] args){  
 G\_O g\_o = **new** G\_O();  
 g\_o.setT(g\_o);  
 g\_o.**D**=g\_o.getD(g\_o);  
 **if**(g\_o.**D**>=1.0/2) {  
 System.***out***.println(**"参数估计无解"**+g\_o.**D**);  
 **return**;  
 }  
 **else if**(g\_o.**D**>0.0 && g\_o.**D**<1.0/2){  
 g\_o.**xl** = (1.0 - 2.0 \* g\_o.**D**) / 2.0;  
 g\_o.**xr** = 1.0 / g\_o.**D**;  
 }  
 **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**;  
 **continue**;  
 }  
 **else if**(g\_o.**f**<-g\_o.**v**){  
 g\_o.**xr**=g\_o.**xm**;  
 **continue**;  
 }  
 **else break**;  
 }  
 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**);  
 **return**;  
 }  
}

## 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

当=0.1时

7@YSKZ3S~__52%]ADX(TB4Q

当=0.01时

PT@N`@}AF`NT6NSHVEZFR)9

当=0.001时

[Z]5NQA0@{V%[E}ZJ1O59VX

当=0.0001时

E4JQY6[P9@BII2J{CB}KWR5

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：

G-O模型的预计的概率密度为：



表示预计模型A的预计概率密度，表示预计模型B的预计概率密度

如果，，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，啧说明预计模型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

D{BLYB0W(96(4N9_B)E`BTI

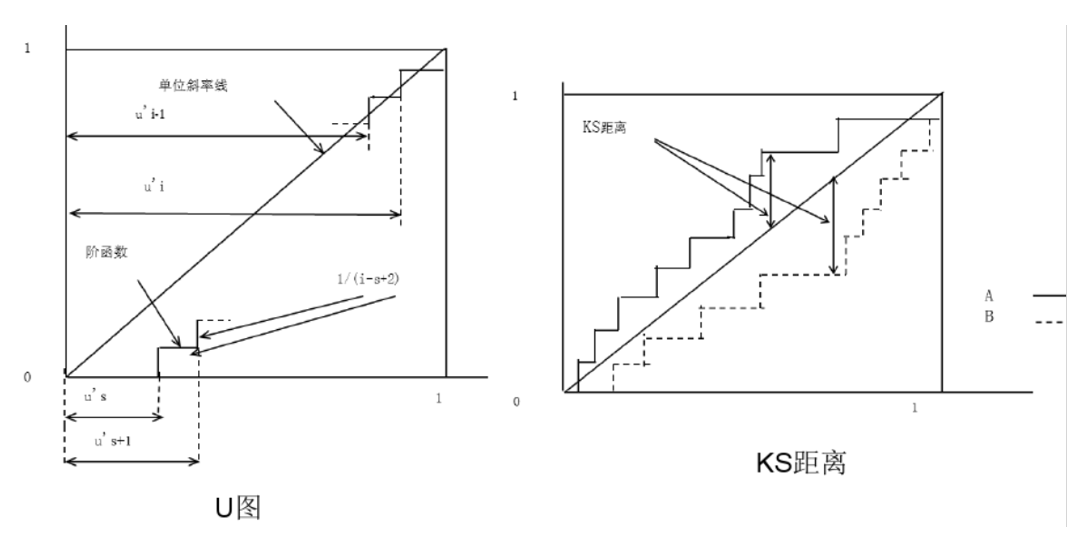
 PLR的值趋近于0，所以G-O模型更优。

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

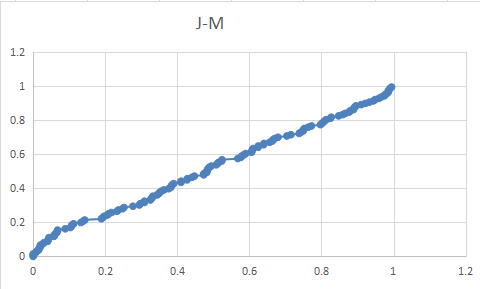
U图的目的是用来判断预测预计分布函数是否均匀地接近于实际分布

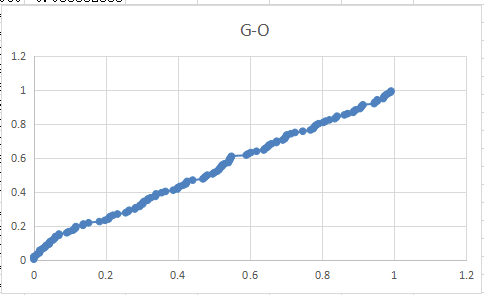
U图就是序列{}的抽样分布函数图，用来表示{}的抽样累积分布函数接近U(0,1)的累积分布函数的程度



J-M模型的预计分布函数为：

G-O模型的预计分布函数为：





求得KS距离

6}U4F9[Z00VU[44]AR7[0ZI

F5L(8Z${Q}W2}IPNHKRBFN5

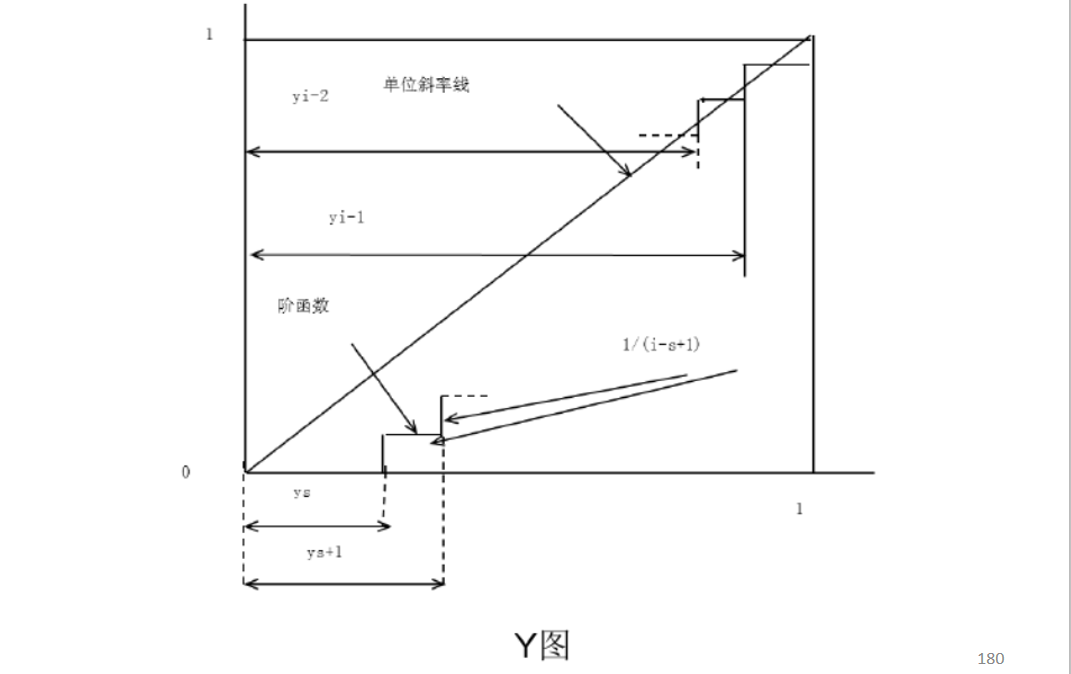
Y图

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图示通过序列进行变换后绘制的。

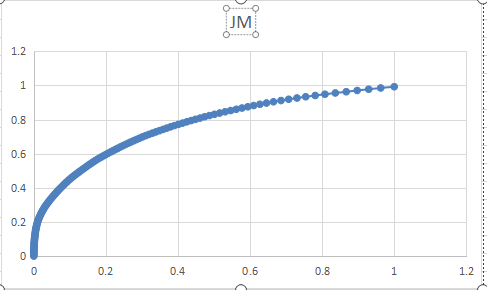
Y图对{}作如下的进一步变换

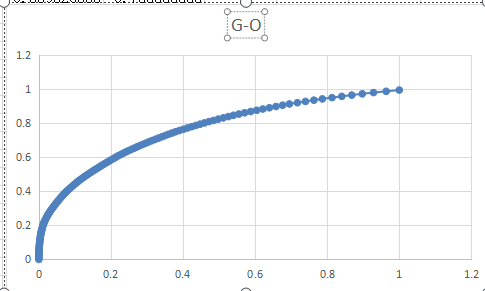
 

1. 图表示{}的样本累积分布函数与45度线的接近程度



拟合曲线可得





求得KS距离

Q_0(O{KYQ{PAFG{I8J_]@KB

$R_KL2)QZDGW)6$G@L)XBJO

可看出GO更加好一点