Ronald Christensen
Department of Mathematics and Statistics
University of New Mexico

R Commands for – L*og-Linear Models and Logistic Regression*, Third Edition

Springer

Preface

This online book is an R companion to *Log-linear Models and Logistic Regression*, Third Edition (*LOGLIN3*). This book presupposes that the reader is already familiar with downloading R, plotting data, reading data files, transforming data, basic housekeeping, loading R packages, and specifying basic linear models. That is the material in Chapters 1 and 3 of my *R Commands for – Analysis of Variance, Design, and Regression: Linear Modeling of Unbalanced Data* (*R Code for ANREG-II*) which is available at http://www.stat.unm.edu/~fletcher/Rcode.pdf. Much of the material here has just been modified/copied from the other volume (but placed appropriately for *LOGLIN3*). A tool that I have found very useful for writing R code is Tom Short's "R Reference Card," http://cran.r-project.org/doc/contrib/Short-refcard.pdf.

Like all of the other R code documents for my books, this is arranged to correspond to the actual book. Thus the R code for performing the things in Chapter 1 of *LOGLIN3* is contained in Chapter 1 of this book, etc. When using this book, all the data can be obtained from http://www.stat.unm.edu/~fletcher/LLM/DATA and are also available at http://stat.unm.edu/~fletcher/llm_data.zip. The code can be obtained from http://www.stat.unm.edu/~fletcher/LLM/Code. Programs listed here as **Code** x.y.x are filed as `Codex-y-z.txt`. *If you are copying R code from a pdf file* into R, "tilde", i.e.,

```
~
```

will often copy incorrectly so that **you may need to delete the copied version of tilde and retype it**. This may also be true for other characters like "caret" ^.

At least on my computer it has become more difficult to install R packages of late, so it is easier to install them all at once. This issue is discussed more in Section 1.6 of R code for ANREG-II. The packages (currently) used in this document are:

```
install.packages("ca")
install.packages("ggplot2")
install.packages("leaps")
install.packages("MASS")
install.packages("psych")
```

```
install.packages("R2OpenBUGS")
install.packages("R2jags")
```

Packages currently discussed but not used are

```
install.packages("bestglm")
install.packages("clogit")
install.packages("exact2x2")
install.packages("exactLoglinTest") (CRAN has removed this.)
install.packages("glmulti")
install.packages("gnm")
install.packages("kequate")
install.packages("logmult")
install.packages("MCMCpack")
install.packages("MSCquartets")
install.packages("nnet")
install.packages("R2WinBUGS")
install.packages("RTDE")
install.packages("StepReg")
install.packages("survey")
```

Contents

Preface vii

Table of Contents ix

1 Introduction 1
- 1.1 Conditional Probability and Independence 1
- 1.2 Random Variables and Expectations 1
- 1.3 The Binomial Distribution 1
- 1.4 The Multinomial Distribution 1
  - 1.4.1 Product-Multinomial Distributions 2
- 1.5 The Poisson Distribution 2

2 Two-Dimensional Tables and Simple Logistic Regression 3
- 2.1 Two Independent Binomials 3
- 2.2 Testing Independence in a 2×2 Table 4
- 2.3 $I \times J$ Tables 4
- 2.4 Maximum Likelihood Theory for Two-Dimensional Tables 5
- 2.5 Log-Linear Models for Two-Dimensional Tables 5
- 2.6 Simple Logistic Regression 7
- 2.7 Exercises 8

3 Three-Dimensional Tables 11
- 3.1 Simpson's Paradox and the Need for Higher-Dimensional Tables . . . 11
- 3.2 Independence and Odds Ratio Models 11
  - 3.2.1 The Model of Complete Independence 11
  - 3.2.2 Models with One Factor Independent of the Other Two 12
  - 3.2.3 Models of Conditional Independence 12
  - 3.2.4 A Final Model for Three-Way Tables 13
- 3.3 Iterative Computation of Estimates 13
- 3.4 Log-Linear Models for Three-Dimensional Tables 14
  - 3.4.1 Estimation 14

3.4.2 Testing Models 14
3.5 Product-Multinomial and Other Sampling Plans 16
3.5.1 Other Sampling Models 16
3.6 Model Selection Criteria 16
3.7 Higher-Dimensional Tables 18
3.8 Exercises 19

4 Logistic Regression 21
4.1 Multiple Logistic Regression 21
4.2 Measuring Model Fit 23
4.3 Logistic Regression Diagnostics 23
4.4 Model Selection Methods 24
4.4.1 Stepwise logistic regression 24
4.4.2 Best subset logistic regression 25
4.5 ANOVA Type Logit Models 26
4.6 Logit Models For a Multinomial Response 30
4.7 Logistic Discrimination and Allocation 33
4.8 Exercises 35

5 Independence Relationships and Graphical Models 37
5.1 Model Interpretations 37
5.2 Graphical and Decomposable Models 37
5.3 Collapsing Tables 37
5.4 Recursive Causal Models 37
5.5 Exercises 37

6 Model Selection Methods and Model Evaluation 39
6.1 Stepwise Procedures for Model Selection 39
6.2 Initial Models for Selection Methods 39
6.2.1 All s-Factor Effects 39
6.2.2 Examining Each Term Individually 40
6.2.3 Tests of Marginal and Partial Association 40
6.2.4 Testing Each Term Last 40
6.3 Example of Stepwise Methods 42
6.3.1 Forward Selection 42
6.3.2 Backward Elimination 42
6.4 Aitkin's Method of Backward Selection 43
6.5 Model Selection Among Decomposable and Graphical Models 44
6.6 Use of Model Selection Criteria 44
6.7 Residuals and Influential Observations 45
6.8 Drawing Conclusions 46
6.9 Exercises 46

7 Models for Factors with Quantitative Levels 47
7.1 Models for Two-Factor Tables 47
7.2 Higher-Dimensional Tables 48
7.3 Unknown Factor Scores 49
7.4 Logit Models with Unknown Scores 50
7.5 Exercises 50

8 Fixed and Random Zeros 51
8.1 Fixed Zeros 51
8.2 Partitioning Polytomous Variables 52
8.3 Random Zeros 54
8.4 Exercises 57

9 Generalized Linear Models 59
9.1 Distributions for Generalized Linear Models 59
9.2 Estimation of Linear Parameters 59
9.3 Estimation of Dispersion and Model Fitting 59
9.4 Summary and Discussion 59
9.5 Exercises 59

10 The Matrix Approach to Log-Linear Models 61
10.1 Maximum Likelihood Theory for Multinomial Sampling 61
10.2 Asymptotic Results 61
10.3 Product-Multinomial Sampling 62
10.4 Inference for Model Parameters 62
10.5 Methods for Finding Maximum Likelihood Estimates 62
10.6 Regression Analysis of Categorical Data 63
10.7 Residual Analysis and Outliers 63
10.8 Exercises 63

11 The Matrix Approach to Logit Models 65
11.1 Estimation and Testing for Logistic Models 65
11.2 Model Selection Criteria for Logistic Regression 65
11.3 Likelihood Equations and Newton-Raphson 65
11.4 Weighted Least Squares for Logit Models 65
11.5 Multinomial Response Models 65
11.6 Asymptotic Results 65
11.7 Discrimination, Allocations, and Retrospective Data 65
11.8 Exercises 65

12 Maximum Likelihood Theory for Log-Linear Models 67
12.1 Notation 67
12.2 Fixed Sample Size Properties 67
12.3 Asymptotic Properties 67
12.4 Applications 67
12.5 Proofs of Lemma 12.3.2 and Theorem 12.3.8 67

13 Bayesian Binomial Regression 69
13.1 Introduction 70
13.1.1 Alternative Specifications 75
13.2 Bayesian Inference: O-ring Data 75
13.2.1 Specifying the Prior and Approximating the Posterior 75
13.2.2 Predictive Probabilities 95
13.2.3 Inference for Regression Coefficients 99
13.2.4 Inference for LD_α 101
13.22Bayesian Inference: Trauma Data 101
13.22.1Specifying the Prior and Approximating the Posterior 103
13.22.2Predictive Probabilities 109
13.22.3Inference for Regression Coefficients 112
13.3 Diagnostics 112
13.3.1 Case Deletion Influence Measures 112
13.3.2 Estimative and Predictive Influence: O-rings 113
13.3.3 Estimative and Predictive Influence: Trauma 115
13.3.4 Model Checking 118
13.3.5 Link Selection 122
13.3.6 Sensitivity Analysis 126
13.4 Posterior Computations 132
13.5 A Log-Linear Model with Over Dispersion (Random Effects) 132
13.5.1 Contingency Tables 135
13.6 OpenBUGS GUI 137

14 Exact Conditional Tests 145
14.1 Two-Factor Tables 145
14.2 Three-Factor Tables 147
14.2.1 Testing $[AC][BC]$ 147
14.2.2 Testing $[B][AC]$ 148
14.3 General Theory 151
14.4 Model Testing 152
14.5 Notes 154

15 Correspondence Analysis 155
15.1 Introduction 155
15.2 Singular Value Decomposition Plot 155
15.3 Correspondence Analysis Plot 155
15.3.1 Nobel Prize Winners 158
15.4 Multiple correspondence analysis 159

16 Polya Trees 161
16.0.1 Alas 161

17 Pythagorian Theorem 165

References 167

Index 169

Chapter 1
Introduction

1.1 Conditional Probability and Independence

1.2 Random Variables and Expectations

1.3 The Binomial Distribution

To evaluate Bin(N,p) densities, use `dbinom(x,N,p)`. The cdf $F(u)$ can be evaluated as `pbinom(u,N,p)` where `p` in `pbinom` stands for probability.

1.4 The Multinomial Distribution

To evaluate Mult(N,p) densities for a vector `p`, at some vector of allowable scalars `x`, use `dmultinom(x,N,p)`. The cdf $F(u)$ can be evaluated as `pmultnom(u,N,p)` where `p` in `pmultinom` stands for probability.

The probability for the table given in this section is given by

```
p=c(.12,.12,.04,.12,.18,.18,.06,.18)
x=c(5,7,4,6,8,7,3,10)
N=50
dmultinom(x,N,p)
```

This number is used in the 3rd edition (0.000002) but does not agree with the number in the 2nd edition (0.000007). I suspect I computed the 1997 book value on a hand calculator canceling many of the terms in the factorials. The following code, that I wrote with numerical stability in mind, does something similar and agrees with `dmultinom`

Code 1.4.1.

```
a=c(50,47,46,11,43,42,41,39,
```

```
38,37,34,33,31,29,
28,3,26,25,24,23,22,21,19,17,15,14,13,11)
b=c(.12^2,.12^2,.12^2,.12^2,.12^2,.12^2,.12^2,.12,
.12,.12,.12,.04,.04,.04,.04, .06,.06,.06,.18^3,
.18^3,.18^3,.18^3,.18^3,.18^3,.18^3,.18^2,.18,.18)
c=a*b
prod(c)
```

1.4.1 Product-Multinomial Distributions

The probability for the table in this section can be computed as

Code 1.4.2.

```
p1=c(.3,.3,.1,.3)
p2=c(.3,.3,.1,.3)
x1=c(10,10,2,8)
x2=c(5,8,1,6)
N1=30
N2=20
dmultinom(x1,N1,p1)*dmultinom(x2,N2,p2)
```

Again, `dmultinom(x1,N1,p1)` agreed with a numerically stable computation but disagreed with what was in the second edition, so I revised the probability in the book.

1.5 The Poisson Distribution

To evaluate Pois(λ) densities, use `dpois(x,lambda)`. The cdf $F(u)$ can be evaluated as `ppois(u,lambda)` where the first `p` in `ppois` stands for probability.

Chapter 2
Two-Dimensional Tables and Simple Logistic Regression

2.1 Two Independent Binomials

A data file might contain three columns: supports, opposes, and the total number surveyed but for this 2×2 table the simplest way to proceed is to just type in the data.

Code 2.1.1.

```
Support=c(309,319)
Oppose=c(191,281)
Total=Support+Oppose
prop.test(Support,Total,correct=FALSE)
```

The test statistic produced is the square of the test statistic in the book.

An alternative way to enter the data is to create a matrix of the admissions and rejections.

Code 2.1.2.

```
OP <- matrix(c(Support,Oppose),ncol=2)
OP
prop.test(OP,correct=FALSE)
```

We could replace `prop.test` with `chisq.test` (using the same arguments) and get the same test but slightly different output and options. The `chisq.test` procedure provides access to Pearson residuals and estimated expected values, things that `prop.test` does not give.

Code 2.1.3.

```
fit <- chisq.test(OP,correct=FALSE)
fit
fit$expected
fit$residual
```

2.2 Testing Independence in a 2×2 Table

Although the sampling scheme differs from the previous section, so the theory is different, the computations are exactly the same.

Code 2.2.1.

```
A=c(483,1101)
B=c(477,1121)
EX <- matrix(c(A,B),ncol=2)
EX
fit <- chisq.test(EX,correct=FALSE)
fit
fit$expected
fit$residual
```

2.3 $I \times J$ Tables

EXAMPLE 2.3.0. With a table this small it is easy to type in the data values.

Code 2.3.0.

```
icu=c(3,36,29,30)
died=c(0,6,5,33)
oth=c(84,1069,228,286)
IJ <- matrix(c(icu,died,oth),ncol=3)
IJ
fit <- chisq.test(IJ,correct=FALSE)
fit
fit$expected
fit$residual
```

EXAMPLE 2.3.1. With a table this small it is easy to type in the data values.

Code 2.3.1.

```
E=c(21,3,7)
G=c(11,2,1)
F=c(4,2,1)
IJ <- matrix(c(E,G,F),ncol=3)
IJ
fit <- chisq.test(IJ,correct=FALSE)
fit
fit$expected
fit$residual
```

2.4 Maximum Likelihood Theory for Two-Dimensional Tables

2.5 Log-Linear Models for Two-Dimensional Tables

The only computing really done in this section is finding Figure 2.1. We begin with the figure but we then fit the data as we previously have in this chapter and finally fit the data using a log-linear model in `glm`. You should examine the output from the two programs for fitting the data to identify that the fitted values and Pearson residuals are identical.

Plot the figure.

Code 2.5.1.

```
rm(list = ls())
cnt=c(34,31,19,23,61,19,23,39,16,17,16,12)
pa=c(1,1,1,1,2,2,2,2,3,3,3,3)
clg=c(1,2,3,4,1,2,3,4,1,2,3,4)

LGCNT=matrix(log(cnt),nrow=4)
LGCNT
test=c(1,2,3)
par(mfrow=c(1,1))
plot(test,LGCNT[1,],type="n",ylab="log(n)",ylim=c(2.5,4.25),
xaxt="n",xlab="Political Affiliation",lty=1,lwd=2)
axis(1,at=c(1,2,3),labels=c("Rep.","Dem.","Ind."))
lines(test,LGCNT[1,],type="o",lty=1,lwd=2)
lines(test,LGCNT[2,],type="o",lty=2,lwd=2)
lines(test,LGCNT[3,],type="o",lty=3,lwd=2)
lines(test,LGCNT[4,],type="o",lty=4,lwd=2)
legend("topright",c("College","Letters",
"Engin.","Agri.","Educ."),lty=c(NA,1,2,3,4))
```

This is how we have previously fitted two-way tables in this chapter.

Code 2.5.2.

```
Rep=c(34,31,19,23)
Dem=c(61,19,23,39)
Ind=c(16,17,16,12)
CLG <- matrix(c(Rep,Dem,Ind),ncol=3)
CLG
fit <- chisq.test(CLG,correct=FALSE)
fit
fit$expected
fit$residual
```

This code fits an equivalent log-linear model. The count data are in one string `cnt` with two other stings to identify the count's political affiliation `pa` and col-

lege `clg`. The likelihood ratio test statistic G^2 (deviance) is listed as the "residual deviance." The data n, fitted values $\hat{m}$, and Pearson residuals are listed in a table along with two things that the book does not introduce for some time, standardized residuals and Cook's distances. The code ends by illustrating a few useful (self-explanitory once you see them) commands.

Code 2.5.3.

```
rm(list = ls())
cnt=c(34,31,19,23,61,19,23,39,16,17,16,12)
pa=c(1,1,1,1,2,2,2,2,3,3,3,3)
clg=c(1,2,3,4,1,2,3,4,1,2,3,4)

# R assumes pa and clg contain real numbers.  For glm,
# we need to specify them as integers using "factor".
PA=factor(pa)
CLG=factor(clg)

# Run glm and obtain summary tables
ts <- glm(cnt ~ PA + CLG,family = poisson)
tsp=summary(ts)
tsp
anova(ts)

# Constructing additional output
rpearson=(cnt-ts$fit)/(ts$fit)^(.5)
# or rpearson = residuals(ts,type="pearson")
rstand=rpearson/(1-hatvalues(ts))^(.5)
infv = c(cnt,ts$fit,hatvalues(ts),rpearson,
  rstand,cooks.distance(ts))
inf=matrix(infv,I(tsp$df[1]+tsp$df[2]),6,dimnames =
list(NULL,c("n","mhat","lev","Pearson","Stand.","C")))
inf

fitted(ts)
df.residual(ts)
sum(residuals(ts,type="pearson")^2)
deviance(ts)
residuals(ts,type="pearson")
```

The command `residuals(ts)` gives something called deviance residuals which are the signed square roots of the components of the G^2 statistic. The command `residuals(ts,type="response")` gives the unstandardized residuals `n - mhat`. Yet another version of residuals are from the last stage of the Newton-Raphson algorithm: `residuals(ts,type="working") = ts$residuals`. Similarly, `ts$fit = fitted(ts)`.

2.6 Simple Logistic Regression

This code uses the logistic/logit model in glm and also includes the computation of diagnostic quantities that are not discussed in Chapter 2. It computes two R^2 values. One (R2) is the squared correlation between the observations and predicted values rather than one based on G^2 statistics and the other (altR2) is based on using deviances like error summs of squares in linear models.

Code 2.6.1.

```
rm(list = ls())
oring.sllr <- read.table(
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-3.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\tab2-1.dat",
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB2-1.dat"),
  sep="",col.names=c("Case","Flt","y","s","f","no"))

attach(oring.sllr)
oring.sllr
#summary(oring.sllr)

#Summary tables
or <- glm(y ~ x,family = binomial)
orp=summary(or)
orp
anova(or)


#prediction
new = data.frame(x=c(31,53))
predict(or,new,type="response")
rpearson=(y-or$fit)/(or$fit*(1-or$fit))^(.5)
rstand=rpearson/(1-hatvalues(or))^(.5)
infv = c(y,or$fit,hatvalues(or),rpearson,
  rstand,cooks.distance(or))
inf=matrix(infv,I(orp$df[1]+orp$df[2]),6,dimnames =
list(NULL,c("y","yhat","lev","Pearson","Stand.","C")))
inf
R2 = (cor(y,or$fit))^2
R2
altR2=(or$null.deviance - or$deviance)/or$null.deviance
altR2
```

We now repeat the logistic regression computations by fitting a corresponding log-linear model in glm.

Code 2.6.2.

```
rm(list = ls())
oring.sllr <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB2-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-3.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\tab2-1.dat",
  sep="",col.names=c("Case","Flt","f","s","x","no"))

attach(oring.sllr)
oring.sllr
#summary(oring.sllr)

# Construct data for log-linear model
# Sting out the failures followed by successes
cnt=c(f,s)
# The temp for each element of cnt
xx=c(x,x)
# The row of the table for each element of cnt
row=c(Case,Case)
# The col. of the table for each element of cnt
# For binary data, f+s=1
# first 23 obs. are first col, 2nd 23 are 2nd col.
col=c(f+s,2*(f+s))
# check that the table is correct
matrix(c(cnt,row,col),ncol=3)

# Fit log-linear model
R=factor(row)
C=factor(col)
fit=glm(cnt ~ R + C + C:xx, family=poisson)
summary(fit)
anova(fit)
```

Compare the parameter estimates associated with `C` and `C:xx` to the logistic regression output. Also compare G^2s.

2.7 Exercises

EXERCISE 2.7.4. *Partitioning Tables*. To perform Lancaster-Irwin partitioning, you "need" to manipulate the data to create appropriate subtables. You can do that in your favorite editor. In Exercise 8.4.3 and *ANREG-II*, Chapter 21 I discuss performing Lancaster-Irwin partitioning by manipulating the subscripts used to define log-linear models.

EXERCISE 2.7.5. *Fisher's Exact Test*.
Commands for performing this test include `fisher.test` and `exact2x2`. These commands are also discussed in Chapter 14.

EXERCISE 2.7.6. *Yule's Q*.
The data should be a 2×2 matrix of counts which can be inputed into the program `Yule` of the library `psych` as `Yule(data,Y=False)`.

EXERCISE 2.7.7. *Freeman-Tukey Residuals*.
You can compute these yourself, or use `FTres` from the package `kequate`. Alternatively, you should be able to use the packages from the next exercise.

EXERCISE 2.7.8. *Power Divergence Statistics*. Examine the program `powerDivStat` from the package `MSCquartets` or the program `MDPD` from the package `RTDE`.

EXERCISE 2.7.10. *Testing for Symmetry*.
Use the command `nominalSymmetryTest`.

EXERCISE 2.7.12. *McNemar's Test*.
Input a matrix of count values into `mcnemar.test`.

Chapter 3
Three-Dimensional Tables

For an analysis of Example 3.0.1, see Example 10.2.6.

3.1 Simpson's Paradox and the Need for Higher-Dimensional Tables

3.2 Independence and Odds Ratio Models

Although log-linear models are not introduced until the next section, we use software for fitting them now.

3.2.1 The Model of Complete Independence

EXAMPLE 3.2.1.

Code 3.2.1.

```
cnt=c(716,79,207,25,819,67,186,22)
ii=c(1,1,1,1,2,2,2,2)
kk=c(1,2,1,2,1,2,1,2)
jj=c(1,1,2,2,1,1,2,2)
II=factor(ii)
JJ=factor(jj)
KK=factor(kk)
sv <- glm(cnt ~ II+JJ+KK,family = poisson)

fitted(sv)
```

```
sum(residuals(sv,type="pearson")^2)
deviance(sv)
df.residual(sv)
residuals(sv,type="pearson")
```

You might also be interested in the output from `summary(sv)` and `anova(sv)`, both here and elsewhere.

3.2.2 Models with One Factor Independent of the Other Two

EXAMPLE 3.2.2.

Code 3.2.2.

```
cnt=c(16,7,15,34,5,3,1,1,3,8,1,3)
ii=c(1,1,1,1,1,1,2,2,2,2,2,2)
jj=c(1,2,1,2,1,2,1,2,1,2,1,2)
kk=c(1,1,2,2,3,3,1,1,2,2,3,3)
II=factor(ii)
JJ=factor(jj)
KK=factor(kk)
sv <- glm(cnt ~ II+JJ:KK,family = poisson)

fitted(sv)
sum(residuals(sv,type="pearson")^2)
deviance(sv)
df.residual(sv)
qchisq(.95,5)
```

For more of an analysis of Example 3.2.2, also see Example 10.2.6.

3.2.3 Models of Conditional Independence

EXAMPLE 3.2.3.

Code 3.2.3.

```
cnt=c(716,79,207,25,819,67,186,22)
ii=c(1,1,1,1,2,2,2,2)
kk=c(1,2,1,2,1,2,1,2)
jj=c(1,1,2,2,1,1,2,2)
II=factor(ii)
```

```
JJ=factor(jj)
KK=factor(kk)
sv <- glm(cnt ~ II:JJ+II:KK,family = poisson)

fitted(sv)
sum(residuals(sv,type="pearson")^2)
deviance(sv)
df.residual(sv)
```

3.2.4 A Final Model for Three-Way Tables

EXAMPLE 3.2.4.

Code 3.2.4.

```
cnt=c(350,150,60,112,26,23,19,80)
ii=c(1,1,1,1,2,2,2,2)
jj=c(1,2,1,2,1,2,1,2)
kk=c(1,1,2,2,1,1,2,2)
II=factor(ii)
JJ=factor(jj)
KK=factor(kk)
sv <- glm(cnt ~ II:JJ+II:KK+JJ:KK,family = poisson)

fitted(sv)
df.residual(sv)
sum(residuals(sv,type="pearson")^2)
deviance(sv)
```

For further analysis of Example 3.2.4, see Example 10.2.4.

3.3 Iterative Computation of Estimates

The generalized linear model procedure `glm` uses Newton-Raphson (iteratively reweighted least squares). [The output refers to it as *Fisher Scoring*.]

To use iterative proportional fitting replace the `glm` command with the `MASS` package command `loglm`. In the `loglm` command there is no need to specify a `family`.

```
library(MASS)
myout = loglm(y ~ model)
myout
myout$df
```

Here the command line `myout` gives a convenient summary. The other post-`glm` commands from the previous section all still work (with different forms for the output) except `df.residual(myout)` needs to be replaced by `myout$df`.

`loglm` employs a somewhat cruder program `loglin` which is not to be confused with a completely different program `LogLin`. `loglin` does not accept a model as input but rather requires the marginal totals that are to be fitted from the contingency table. Apparently, `loglm` uses the model to figure out the appropriate marginal totals. As such, `loglm` seems to work only with ANOVA type models (even though iterative proportional fitting can be made to work for arbitrary log-linear models). In particular, it does not work for ACOVA type models. Quantitative effects seem to be treated as factors and, although the output from the command `myout` seems to give the correct degrees of freedom for treating the quantitative effect as a factor, using `anova(myout)` seems to have big problems with getting the degrees of freedom correct

3.4 Log-Linear Models for Three-Dimensional Tables

3.4.1 Estimation

The discussion in the book focuses on estimation based on the estimated expected cell counts. `glm` uses the `summary()` command to produce estimates of the actual model parameters. These can be quite complicated to use but the pattern of their use is analogous to their use with linear models which is explained in *ANREG-II* and its related computing document.

3.4.2 Testing Models

We now add the `anova` command to our fitting including its use for comparing models.

EXAMPLE 3.4.1. The last residual deviances are what we want.

Code 3.4.1.

```
cnt=c(16,7,15,34,5,3,1,1,3,8,1,3)
ii=c(1,1,1,1,1,1,2,2,2,2,2,2)
jj=c(1,2,1,2,1,2,1,2,1,2,1,2)
kk=c(1,1,2,2,3,3,1,1,2,2,3,3)
II=factor(ii)
JJ=factor(jj)
KK=factor(kk)
sv <- glm(cnt ~ II+JJ+KK+JJ:KK,family = poisson)
```

```
anova(sv)
```

EXAMPLE 3.4.2.

Code 3.4.2.

```
cnt=c(716,79,207,25,819,67,186,22)
ii=c(1,1,1,1,2,2,2,2)
kk=c(1,2,1,2,1,2,1,2)
jj=c(1,1,2,2,1,1,2,2)
II=factor(ii)
JJ=factor(jj)
KK=factor(kk)
sv7 <- glm(cnt ~ II:JJ+II:KK+JJ:KK,family = poisson)
sv6 <- glm(cnt ~ II:JJ+II:KK,family = poisson)
sv5 <- glm(cnt ~ II:JJ+JJ:KK,family = poisson)
sv4 <- glm(cnt ~ II:KK+JJ:KK,family = poisson)
sv1 <- glm(cnt ~ II+JJ:KK,family = poisson)
sv2 <- glm(cnt ~ JJ+II:KK,family = poisson)
sv3 <- glm(cnt ~ II:JJ+KK,family = poisson)
sv0 <- glm(cnt ~ II+JJ+KK,family = poisson)


tab7=c(7,df.residual(sv7),
sum(residuals(sv7,type="pearson")^2),
deviance(sv7),1-pchisq(deviance(sv7),df.residual(sv7)))
tab6=c(6,df.residual(sv6),
sum(residuals(sv6,type="pearson")^2),
deviance(sv6),1-pchisq(deviance(sv6),df.residual(sv6)))
tab5=c(5,df.residual(sv5),
sum(residuals(sv5,type="pearson")^2),
deviance(sv5),1-pchisq(deviance(sv5),df.residual(sv5)))
tab4=c(4,df.residual(sv4),
sum(residuals(sv4,type="pearson")^2),
deviance(sv4),1-pchisq(deviance(sv4),df.residual(sv4)))
tab1=c(1,df.residual(sv1),
sum(residuals(sv1,type="pearson")^2),
deviance(sv1),1-pchisq(deviance(sv1),df.residual(sv1)))
tab2=c(2,df.residual(sv2),
sum(residuals(sv2,type="pearson")^2),
deviance(sv2),1-pchisq(deviance(sv2),df.residual(sv2)))
tab3=c(3,df.residual(sv3),
sum(residuals(sv3,type="pearson")^2),
deviance(sv3),1-pchisq(deviance(sv3),df.residual(sv3)))
tab0=c(0,df.residual(sv0),
sum(residuals(sv0,type="pearson")^2),
```

```
deviance(sv0),1-pchisq(deviance(sv0),df.residual(sv0)))

t(matrix(c(tab7,tab6,tab5,tab4,tab1,tab2,tab3,tab0),5,8))
anova(sv0,sv6)
qchisq(0.95,2)
anova(sv3,sv6)
qchisq(0.95,1)
anova(sv3,sv7)
```

3.5 Product-Multinomial and Other Sampling Plans

3.5.1 Other Sampling Models

For performing exact tests, the package https://cran.r-project.org/web/packages/exactLoglinTest/index.html was developed but seems to have been removed from CRAN for not fixing reported problems. Other software is reported for Exercise 2.7.5 on Fisher's Exact Test.

For survey data the key R package seems to be `survey` which includes a program `svyglm` that is similar to `glm`. UCLA Survey Data Analysis with R nicely explicates the use of `survey` with the later material being more relevant to our tasks.

3.6 Model Selection Criteria

When the book was originally written, software did not readily compute AIC so $A-q \equiv$ `AICq` was used because it was easy to compute by hand from the output df and G^2. In R, for a fitted model, say `svm`, the computation below is `AICq=deviance(svm)-2*df.residual(svm)`. Now AIC is part of R's standard output and can be manipulated as `AIC(svm)`. **The key point to notice is that, for various models, differences between R's AIC agree with the corresponding differences between $A-q$,** so they lead to the same ordering of models.

Code 3.6.1.

```
cnt=c(716,79,207,25,819,67,186,22)
ii=c(1,1,1,1,2,2,2,2)
kk=c(1,2,1,2,1,2,1,2)
jj=c(1,1,2,2,1,1,2,2)
II=factor(ii)
JJ=factor(jj)
KK=factor(kk)
sv7 <- glm(cnt ~ II:JJ+II:KK+JJ:KK,family = poisson)
```

```
sv6 <- glm(cnt ~ II:JJ+II:KK,family = poisson)
sv5 <- glm(cnt ~ II:JJ+JJ:KK,family = poisson)
sv4 <- glm(cnt ~ II:KK+JJ:KK,family = poisson)
sv1 <- glm(cnt ~ II+JJ:KK,family = poisson)
sv2 <- glm(cnt ~ JJ+II:KK,family = poisson)
sv3 <- glm(cnt ~ II:JJ+KK,family = poisson)
sv0 <- glm(cnt ~ II+JJ+KK,family = poisson)


tab7=c(7,df.residual(sv7), deviance(sv7),
deviance(sv7)-2*df.residual(sv7),
((deviance(sv0)-deviance(sv7))/(deviance(sv0))),
(1-(deviance(sv7)*df.residual(sv0)/
(deviance(sv0)*df.residual(sv7))))  )
tab6=c(6,df.residual(sv6), deviance(sv6),
deviance(sv6)-2*df.residual(sv6),
((deviance(sv0)-deviance(sv6))/(deviance(sv0))),
(1-(deviance(sv6)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv6))))  )
tab5=c(5,df.residual(sv5), deviance(sv5),
deviance(sv5)-2*df.residual(sv5),
((deviance(sv0)-deviance(sv5))/(deviance(sv0))),
(1-(deviance(sv5)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv5))))  )
tab4=c(4,df.residual(sv4), deviance(sv4),
deviance(sv4)-2*df.residual(sv4),
((deviance(sv0)-deviance(sv4))/(deviance(sv0))),
(1-(deviance(sv4)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv4))))  )
tab1=c(1,df.residual(sv1), deviance(sv1),
deviance(sv1)-2*df.residual(sv1),
((deviance(sv0)-deviance(sv1))/(deviance(sv0))),
(1-(deviance(sv1)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv1))))  )
tab2=c(2,df.residual(sv2), deviance(sv2),
deviance(sv2)-2*df.residual(sv2),
((deviance(sv0)-deviance(sv2))/(deviance(sv0))),
(1-(deviance(sv2)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv2))))  )
tab3=c(3,df.residual(sv3), deviance(sv3),
deviance(sv3)-2*df.residual(sv3),
((deviance(sv0)-deviance(sv3))/(deviance(sv0))),
(1-(deviance(sv3)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv3))))  )
tab0=c(0,df.residual(sv0), deviance(sv0),
```

```
deviance(sv0)-2*df.residual(sv0),
((deviance(sv0)-deviance(sv0))/(deviance(sv0))),
(1-(deviance(sv0)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv0))))  )

t(matrix(c(tab7,tab6,tab5,tab4,tab1,tab2,tab3,tab0),6,8))
```

In retrospect, I probably should have created an R function for computing $A-q$. R functions are used in Chapter 13.

3.7 Higher-Dimensional Tables

Muscle tension changes.

Code 3.7.1.

```
rm(list = ls())
tense <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example3-7-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10a.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-7-1.dat",
  sep="",col.names=c("y","Tn","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
M=factor(Ms)
D=factor(Dr)
T=factor(Tn)
m7 <- glm(y ~ T:W:M+T:W:D+T:M:D+W:M:D,family = poisson)
m4 <- glm(y ~ T:W+T:M+T:D+W:M+W:D+M:D,family = poisson)
m0 <- glm(y ~ T + W + M + D,family = poisson)

df=c(m7$df.residual,m4$df.residual,m0$df.residual)
G2=c(m7$deviance,m4$deviance,m0$deviance)
A2q=G2-(2*df)
modelm=c(df,G2,A2q)
model=matrix(modelm,3,3,
      dimnames=list(NULL,c("df","G2","A-q")))
model
```

You can also get the key statistics from the following commands

Code 3.7.2.

```
m7 <- glm(y ~ T*W*M+T*W*D+T*M*D+W*M*D,family=poisson)
m7p=summary(m7)
m7p
anova(m7)
```

What you want is in the last 2 columns. R is fitting the models sequentially, adding in each term of the model.

See Section 4.6 for the Abortion Opinion data.

3.8 Exercises

EXERCISE 3.8.9. *The Mantel-Haenszel Statistic.*
Use `mantelhaen.test`.

Chapter 4
Logistic Regression

The base R program `glm` fits generalized linear models including logistic/logit models. We will focus on that.

Christensen (2015, Chapter 20) discusses some of the specialized features available from some software written specifically for logistic regression. In particular, he has code for the SAS and Minitab logistic regression programs. His corresponding R code is at R-ANREGII

The text does not include discussion of incorporating penalty functions (regularization) as discussed in Christensen (2019, Chapter 13). `glm` does not incorporate this option except through augmenting the data. Support vector machines are an alternative to logistic regression and are also discussed in Christensen (2019, Chapter 13) and its R companion R-ALMIII

4.1 Multiple Logistic Regression

This code includes diagnostic quantities that are not discussed until a few sections later. This fits the full model, the other fitted models are easy.

Code 4.1.1.

```
rm(list = ls())
chap.mlr <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/CHAPMAN.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\chapman.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\chapman.dat",
sep="",col.names=
c("Case","Ag","S","D","Ch","H","W","y"))

attach(chap.mlr)
chap.mlr
#summary(chap.mlr)
```

```
#Summary tables
cm <- glm(y ~ Ag+S+D+Ch+H+W,family = binomial)
cmp=summary(cm)
cmp
#anova(cm)

# Diagnostics
rpearson=(y-cm$fit)/(cm$fit*(1-cm$fit))^(.5)
rstand=rpearson/(1-hatvalues(cm))^(.5)
infv = c(y,cm$fit,hatvalues(cm),rpearson,
  rstand,cooks.distance(cm))
inf=matrix(infv,I(cmp$df[1]+cmp$df[2]),6,dimnames =
list(NULL,c("y","yhat","lev","Pearson","Stand.","C")))
inf

# Tests against Model (1)
cmAg <- glm(y ~ Ag,family = binomial)
anova(cmAg,cm)

#Variations on AIC
# q=400=200*2
Aq=AIC(cmAg)-400
Aq1=deviance(cmAg)-2*df.residual(cmAg)
c(Aq,Aq1)
Astar=258.1+Aq
out=
c(df.residual(cmAg),deviance(cmAg),Aq,Astar,AIC(cmAg))
matrix(out,1,5,dimnames =
list(NULL,c("df","G2","A-2q","A*","AIC")))
```

The rest of the output is just reapplying modifications of the fitting code and applying the formulas

Code for Figure 4.1 follows.

Code 4.1.2.

```
x=seq(20,70,.5)
w =-4.5173+(0.04590*x)+(0.00686*140)+(-0.00694*90)+
  (0.00631*200)+(-0.07400*69)+(0.02014*200)
w1=-4.5173+(0.04590*x)+(0.00686*140)+(-0.00694*90)+
  (0.00631*300)+(-0.07400*69)+(0.02014*200)
y=exp(w)/(1+exp(w))
y1=exp(w1)/(1+exp(w1))
plot(x,y1,type="l",xlim=c(20,70),ylim=c(0,.5),
   ylab="Fitted",xlab="Age",lty=2)
lines(x,y,type="l",lty=1)
```

```
legend("topleft",c("Chol","300","200"),lty=c(NA,2,1))
```

4.2 Measuring Model Fit

This section of the book does not propose a formal test but it is quite similar to the widely programmed Hosmer and Lemshow lack-of-fit test, *which doesn't work* (at least not when compared to a χ^2 as it is usually programmed).

4.3 Logistic Regression Diagnostics

A table of diagnostics was given in Section 1. We can demonstrate the one-step algorithms.

First we give the standard fitting algorithm. Note that this gives slightly different standard errors than the software I used for the book.

Code 4.3.1.

```
rm(list = ls())
chap.mlr <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/CHAPMAN.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\chapman.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\chapman.dat",
sep="",col.names=
c("Case","Ag","S","D","Ch","H","W","y"))

attach(chap.mlr)
chap.mlr
#summary(chap.mlr)

#Summary tables
cm <- glm(y ~ Ag+Ch+W,family = binomial)
cmp=summary(cm)
cmp
#anova(cm)

# Diagnostics
rpearson=(y-cm$fit)/(cm$fit*(1-cm$fit))^(.5)
rstand=rpearson/(1-hatvalues(cm))^(.5)
infv = c(y,cm$fit,hatvalues(cm),rpearson,
  rstand,cooks.distance(cm))
inf=matrix(infv,I(cmp$df[1]+cmp$df[2]),6,dimnames =
list(NULL,c("y","phat","lev","Pearson","Stand.","C")))
```

```
inf
```

Now we construct the one-step model. Remember that this program is only for binary counts. Although there were some slight differences in the glm fit, these agree with the table in the book.

Code 4.3.2.

```
RWT=cm$fit*(1-cm$fit)
Y0=log(cm$fit/(1-cm$fit))
Y=Y0+(y-cm$fit)/RWT
# The following # command should be and is a
# nearly perfect fit.
#summary(lm(Y0 ~ Ag+Ch+W,weight=RWT))
one=lm(Y ~ Ag+Ch+W,weight=RWT)
```

The following gives the leverages and Cook's distances from the one-step procedure and compares them to the values from the glm procedure for the 4 cases discussed in the book.

Code 4.3.3.

```
rtMSE=summary(one)$sigma
levone=hatvalues(one)
cookone=(cooks.distance(one)*rtMSE^2)
c(levone[41],levone[86],levone[126],levone[192])
c(hatvalues(cm)[41],hatvalues(cm)[86],
hatvalues(cm)[126],hatvalues(cm)[192])
c(cookone[41],cookone[86],cookone[126],cookone[192])
c(cooks.distance(cm)[41],cooks.distance(cm)[86],
cooks.distance(cm)[126],cooks.distance(cm)[192])
```

To delete case 41 and refit, use `y[41]=NA` although you might want to do this on a copy of `y` rather than on `y` itself.

4.4 Model Selection Methods

4.4.1 Stepwise logistic regression

This chooses models based on the AIC criterion, so they may be a bit different from the book. As illustrated earlier, read in the data and obtain the `glm` output.

Code 4.4.1.

```
ch = glm(y ~ Ag+S+D+Ch+H+W,family=binomial)
chstep <- step(ch, direction="backward")
chstep
```

Other "directions" include `both` and `forward` but forward requires additional commands, see Section 10.3. You get similar results by replacing the `glm` output in `ch` with the `lm` output from

```
ch1 = lm(yy ~ Ag+S+D+Ch+H+W,weights=rtw)
```

where `yy` and `rtw` are defined in the next subsection. The R package `StepReg` can use *P* values rather than AIC.

4.4.2 Best subset logistic regression

Sources for performing best subset logistic regression include the packages `bestglm` (by A.I. McLeod, Changjiang Xu and Yuanhao Lai) and `glmulti`, cf. Calcagno and de Mazancourt (2010). Both seem to do full, rather than one-step, fits of the models although `glmulti` can use a genetic algorithm. `StepReg` currently uses a method similar to SAS based on Score tests.

The method in the book is a faster approximate method that uses the Leaps and Bounds algorithm used with standard regression. The method starts with the full model and performs only one step of the Newton-Raphson/Iteratively Reweighted Least Squares algorithm to determine the best models. This is a far better procedure than the score test method used by SAS Proc Logistic because it starts from the full model, which should be a good model, rather than the intercept-only model used by the score test.

Code 4.4.2.

```
rm(list = ls())
chap <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/CHAPMAN.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\chapman.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\chapman.dat",
sep="",col.names=
c("Case","Ag","S","D","Ch","H","W","y"))
attach(chap)
chap
summary(chap)

#Summary tables
ch = glm(y ~ Ag+S+D+Ch+H+W,family=binomial)
chp=summary(ch)
chp
#anova(ch)

rwt=ch$fit*(1-ch$fit)
yy=log(ch$fit/(1-ch$fit))+(y-ch$fit)/rwt
```

```
# If Bin(n_i,p_i)s have n_i different from 1,
# multiply rwt and second term in yy by by n_i

ch1 <- lm(yy ~ Ag+S+D+Ch+H+W,weights=rwt)
ch1p=summary(ch1)
ch1p
anova(ch1)
# Note the agreement between the glm and lm fits!!!

# assign number of best models and number of
# predictor variables.

#install.packages("leaps")
library(leaps)
x <- model.matrix(ch1)[,-1]
nb=3
xp=ch1p$df[1]-1
dfe=length(y)- 1- c(rep(1:(xp-1),each=nb),xp)
g <- regsubsets(x,yy,nbest=nb,weights=rwt)
gg = summary(g)
tt=c(gg$rsq,gg$adjr2,gg$cp,sqrt(gg$rss/dfe))
tt1=matrix(tt,nb*(xp-1)+1,4,
dimnames = list(NULL,c("R2","AdjR2","Cp","RootMSE")))
tab1=data.frame(tt1,gg$outmat)
tab1
```

4.5 ANOVA Type Logit Models

The R code below gives slightly different answers than were given in the 1997 version of the book. While I presume the R code is probably more accurate than the book (I presume that software accuracy has improved over the last 25 years), I decided not to change the numbers in the book as an object lesson. Regardless of the program used, the results are from an iterative procedure, and are subject to arbitrary criteria such as what it means for a sequence to have converged. *Getting **slightly** different numbers from different software is to be expected. Substantive conclusions about the data should never be so precise as to depend on such slight differences.*

We now generate Table 4.2.

Code 4.5.1.

```
tense <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/
Example3-7-1-logistic.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10.dat",
```

```
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-7-1-logistic.dat",
  sep="",col.names=c("High","Low","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
M=factor(Ms)
D=factor(Dr)
T=cbind(High,Low)
sv7 <- glm(T ~ W:M+W:D+M:D,family = binomial)
sv6 <- glm(T ~ W:M+W:D,family = binomial)
sv5 <- glm(T ~ W:M+M:D,family = binomial)
sv4 <- glm(T ~ W:D+M:D,family = binomial)
sv1 <- glm(T ~ W+M:D,family = binomial)
sv2 <- glm(T ~ M+W:D,family = binomial)
sv3 <- glm(T ~ W:M+D,family = binomial)
sv0 <- glm(T ~ W+M+D,family = binomial)
svd <- glm(T ~ W+M,family = binomial)
svm <- glm(T ~ W+D,family = binomial)
svw <- glm(T ~ M+D,family = binomial)


tab7=c(7,df.residual(sv7),deviance(sv7),
1-pchisq(deviance(sv7),df.residual(sv7)),
-2*df.residual(sv7)+deviance(sv7))
tab6=c(6,df.residual(sv6),deviance(sv6),
1-pchisq(deviance(sv6),df.residual(sv6)),
-2*df.residual(sv6)+deviance(sv6))
tab5=c(5,df.residual(sv5),deviance(sv5),
1-pchisq(deviance(sv5),df.residual(sv5)),
-2*df.residual(sv5)+deviance(sv5))
tab4=c(4,df.residual(sv4),deviance(sv4),
1-pchisq(deviance(sv4),df.residual(sv4)),
-2*df.residual(sv4)+deviance(sv4))
tab1=c(1,df.residual(sv1),deviance(sv1),
1-pchisq(deviance(sv1),df.residual(sv1)),
-2*df.residual(sv1)+deviance(sv1))
tab2=c(2,df.residual(sv2),deviance(sv2),
1-pchisq(deviance(sv2),df.residual(sv2)),
-2*df.residual(sv2)+deviance(sv2))
tab3=c(3,df.residual(sv3),deviance(sv3),
1-pchisq(deviance(sv3),df.residual(sv3)),
-2*df.residual(sv3)+deviance(sv3))
tab0=c(0,df.residual(sv0),deviance(sv0),
```

```
1-pchisq(deviance(sv0),df.residual(sv0)),
-2*df.residual(sv0)+deviance(sv0))
tabd=c(2,df.residual(svd),deviance(svd),
1-pchisq(deviance(svd),df.residual(svd)),
-2*df.residual(svd)+deviance(svd))
tabm=c(3,df.residual(svm),deviance(svm),
1-pchisq(deviance(svm),df.residual(svm)),
-2*df.residual(svm)+deviance(svm))
tabw=c(0,df.residual(svw),deviance(svw),
1-pchisq(deviance(svw),df.residual(svw)),
-2*df.residual(svw)+deviance(svw))

t(matrix(c(tab7,tab6,tab5,tab4,tab3,
tab2,tab1,tab0,tabd,tabm,tabw),5,11))

anova(sv0,sv6)
qchisq(0.95,2)
anova(sv3,sv6)
qchisq(0.95,1)
anova(sv3,sv7)
```

Generate Tables 4.3 and 4.4.

Code 4.5.2.

```
rm(list = ls())
tense <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example3-7-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10a.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-7-1.dat",
  sep="",col.names=c("y","Tn","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
M=factor(Ms)
D=factor(Dr)
T=factor(Tn)
m6 <- glm(y ~ T:W + T:M:D + W:M:D,family = poisson)
fitted(m6)
c(fitted(m6)[1]/fitted(m6)[9],fitted(m6)[2]/
fitted(m6)[10],fitted(m6)[3]/fitted(m6)[11],
fitted(m6)[4]/fitted(m6)[12],fitted(m6)[5]/
fitted(m6)[13],fitted(m6)[6]/fitted(m6)[14],
fitted(m6)[7]/fitted(m6)[15],fitted(m6)[8]/
fitted(m6)[16],fitted(m6)[1]/fitted(m6)[9])
```

This obtains Table 4.4 directly from the logit model.

Code 4.5.3.

```
rm(list = ls())
tense <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example3-7-1-logistic.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10a.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-7-1-logistic.dat",
  sep="",col.names=c("High","Low","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
M=factor(Ms)
D=factor(Dr)
T=cbind(High,Low)
ts <- glm(T ~ W + M*D,family = binomial)
tsp=summary(ts)
tsp
anova(ts)

fitted(ts)/(1-fitted(ts))
```

This obtains Tables 4.5 and 4.6 directly from the logit model.

Code 4.5.4.

```
rm(list = ls())
tense <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example3-7-1-logistic.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10a.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-7-1-logistic.dat",
  sep="",col.names=c("High","Low","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
M=factor(Ms)
D=factor(Dr)
T=cbind(High,Low)
ts <- glm(T ~ W:M + M*D,family = binomial)
tsp=summary(ts)
tsp
anova(ts)
```

```
fitted(ts)/(1-fitted(ts))
```

4.6 Logit Models For a Multinomial Response

This code analyzes the data in Table 3.1 to obtain Table 4.7.

Code 4.6.1.

```
rm(list = ls())
abt <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB3-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-4.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\ABORT.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TAB3-1.DAT",
  sep="",col.names=c("R","S","A","O","y"))
attach(abt)
abt
#summary(abt)

r=factor(R)
o=factor(O)
s=factor(S)
a=factor(A)

m15 <- glm(y ~ r:s:a+r:s:o+r:o:a+s:o:a ,family = poisson)
m14 <- glm(y ~ r:s:a + r:s:o + r:o:a ,family = poisson)
m13 <- glm(y ~ r:s:a + r:s:o + s:o:a ,family = poisson)
m12 <- glm(y ~ r:s:a + r:o:a + s:o:a ,family = poisson)
m11 <- glm(y ~ r:s:a + r:s:o + o:a   ,family = poisson)
m10 <- glm(y ~ r:s:a + r:o:a + s:o ,family = poisson)
m9 <- glm(y ~ r:s:a + s:o:a + r:o  ,family = poisson)
m8 <- glm(y ~ r:s:a + r:o + s:o + o:a,family = poisson)
m7 <- glm(y ~ r:s:a + r:o + s:o ,family = poisson)
m6 <- glm(y ~ r:s:a + r:o + o:a ,family = poisson)
m5 <- glm(y ~ r:s:a + s:o + o:a ,family = poisson)
m4 <- glm(y ~ r:s:a + r:o   ,family = poisson)
m3 <- glm(y ~ r:s:a + s:o   ,family = poisson)
m2 <- glm(y ~ r:s:a + o:a   ,family = poisson)
m1 <- glm(y ~ r:s:a + o  ,family = poisson)

df=c(m15$df.residual,m14$df.residual,m13$df.residual,
m12$df.residual,m11$df.residual,m10$df.residual,
m9$df.residual,m8$df.residual,
m7$df.residual,m6$df.residual,m5$df.residual,
```

```
m4$df.residual,m3$df.residual,m2$df.residual,
m1$df.residual)
G2=c(m15$deviance,m14$deviance,m13$deviance,
m12$deviance,m11$deviance,m10$deviance,m9$deviance,
m8$deviance,m7$deviance,m6$deviance,m5$deviance,
m4$deviance,m3$deviance,m2$deviance,m1$deviance)

A2q=G2-(2*df)
modelm=c(df,G2,A2q)
model=matrix(modelm,15,3,dimnames =
      list(NULL,c("df","G2","A-q")))
model
```

We now get the output for Table 4.8 of the book. When looking at the estimated expected cell counts and Pearson residuals associated with the next group of commands, it is *important to notice* that in the data file the White Males are listed in a different order than the other race-sex groups.

Code 4.6.2.

```
rm(list = ls())
abt <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB3-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-4.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Tab3-1.dat",
  sep="",col.names=c("R","S","A","O","y"))
attach(abt)
abt
#summary(abt)

r=factor(R)
o=factor(O)
s=factor(S)
a=factor(A)
m11 <- glm(y ~ r:s:a + r:s:o + o:a   ,family = poisson)


m11s=summary(m11)
m11s
anova(m11)

rpearson=(y-m11$fit)/(m11$fit)^(.5)
rstand=rpearson/(1-hatvalues(m11))^(.5)
infv = c(y,m11$fit,hatvalues(m11),rpearson,rstand,
cooks.distance(m11))
inf=matrix(infv,I(m11s$df[1]+m11s$df[2]),6,dimnames =
list(NULL,c("y","yhat","lev","Pearson","Stand.","C")))
```

```
inf
```

To fit log-linear models to the data with "undecided"s eliminated, use the data file `Tab3-1del`. To fit logistic models to either the data with "undecided"s eliminated or to the "decided-undecided" data, use the data file `Tab3-1-logistic`. **It contains column labels that need to be removed before this code will work.** For the "decided-undecided" data you need to construct a new variable for the number of trials, Total2=Total+Und and use that as the weight variable in, say,

```
yy=Total/Total2
abd <- glm(yy~R:S+A,family=binomial,weights=Total2)
```

We now examine fitting the logistic models (4.6.5) through (4.6.7) from the book to the data with "undecideds" eliminated.

Code 4.6.3.

```
rm(list = ls())
abop <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB3-1-logistic.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-15.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\tab3-1-logistic.dat",
sep="",col.names=
c("Case","Race","Sex","Age","Yes","No","Total","Und"))
attach(abop)
abop
#summary(abop)

#Summary tables
R=factor(Race)
S=factor(Sex)
A=factor(Age)
y=Yes/Total
# Model (4.6.5)
ab <- glm(y~R:S+A,family=binomial,weights=Total)
abp=summary(ab)
abp

odds=ab$fit/(1-ab$fit)
odds

# Model (4.6.6)
ab6 <- glm(y~R:S+Age,family=binomial,weights=Total)
abp=summary(ab6)
abp
anova(ab6,ab5)

# Model (4.6.7)
```

```
Men=Race*(Sex-1)
m=factor(Men)
ab7 <- glm(y~m+A,family=binomial,weights=Total)
abp=summary(ab7)
abp
anova(ab7,ab5)

# Model (4.6.8)
ab8 <- glm(y~m+Age,family=binomial,weights=Total)
abp=summary(ab8)
abp
anova(ab8,ab5)

odds=ab8$fit/(1-ab8$fit)
oddstable=matrix(odds,6,4,dimnames =
list(NULL,c("Male", "   White Female",
"      Male","   Nonwhite Female")))
oddstable
```

Also see `multinom` in library `nnet` and `polr` in `MASS`

4.7 Logistic Discrimination and Allocation

The first thing we have to do is create the 3×21 table illustrated in the book. We then fit the model and finally we get the entries for the book's Tables 4.11 and 4.12.

Code 4.7.1.

```
rm(list = ls())
cush <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB4-11.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-11.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TAB4-11.DAT",
  sep="",col.names=c("Syn","Tetra","Preg"))
attach(cush)
cush

#Create a 3 x 21 table of 0-1 entries,
#each row has 1's for a different type of syndrome
j=rep(seq(1,21),3)
i=c(rep(1,21),rep(2,21),rep(3,21))
Tet=c(Tetra,Tetra,Tetra)
Pre=c(Preg,Preg,Preg)
y=c(Syn,Syn,Syn)
y[1:6]=1
```

```
y[7:21]=0
y[22:27]=0
y[28:37]=1
y[38:58]=0
y[59:63]=1
datal=c(y,i,j,Tet,Pre)
datl=matrix(datal,63,5,dimnames =
list(NULL,c("y", "i", "j","Tet","Pre")))
datl

#Fit the log-linear model for logistic discrimination.
i=factor(i)
j=factor(j)
lp=log(Pre)
lt=log(Tet)
ld <- glm(y ~ i + j + i:lt +i:lp ,family = poisson)
ldp=summary(ld)
ldp
anova(ld)

# Table 4.12
q=ld$fit
# Divide by sample sizes
p1=ld$fit[1:21]/6
p2=ld$fit[22:42]/10
p3=ld$fit[43:63]/5
# Produce table
estprob = c(Syn,p1,p2,p3)
EstProb=matrix(estprob,21,4,dimnames =
list(NULL,c("Group", "A", "B","C")))
EstProb

# Table 4.13 Proportional prior probabilities.
post = c(Syn,ld$fit)
PropProb=matrix(post,21,4,dimnames =
list(NULL,c("Group", "A", "B","C")))
PropProb

# Table 4.13 Equal prior probabilities.
p=p1+p2+p3
pp1=p1/p
pp2=p2/p
pp3=p3/p
post = c(Syn,pp1,pp2,pp3)
EqProb=matrix(post,21,4,dimnames=
```

```
list(NULL,c("Group","A","B","C")))
EqProb
```

4.8 Exercises

Chapter 5
Independence Relationships and Graphical Models

There is no computing in this entire chapter.

5.1 Model Interpretations

5.2 Graphical and Decomposable Models

5.3 Collapsing Tables

5.4 Recursive Causal Models

5.5 Exercises

Chapter 6
Model Selection Methods and Model Evaluation

6.1 Stepwise Procedures for Model Selection

No computing.

6.2 Initial Models for Selection Methods

6.2.1 All s-Factor Effects

Code 6.2.1.

```
rm(list = ls())
tense <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example3-7-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10a.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-7-1.dat",
  sep="",col.names=c("y","Tn","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
M=factor(Ms)
D=factor(Dr)
T=factor(Tn)
m7 <- glm(y ~ T:W:M+T:W:D+T:M:D+W:M:D,family=poisson)
m4 <- glm(y ~ T:W+T:M+T:D+W:M+W:D+M:D,family=poisson)
m0 <- glm(y ~ T + W + M + D,family = poisson)

df=c(m7$df.residual,m4$df.residual,m0$df.residual)
```

```
G2=c(m7$deviance,m4$deviance,m0$deviance)
A2q=G2-(2*df)
modelm=c(df,G2,A2q)
model=matrix(modelm,3,3,dimnames=
list(NULL,c("df","G2","A-q")))
model
```

6.2.2 Examining Each Term Individually

No computing.

6.2.3 Tests of Marginal and Partial Association

The computations are simple but repetitive. The problem is identifying the models you need to fit. The beauty of BMDP 4F is that it did these for you automatically. I am not about to write a front end that determines all of these.

Apparently, some versions of SAS prior to SAS 9.4M2 (2014) allowed you to run BMDP procedures, https://v8doc.sas.com/sashtml/unixc/z0397594.htm. Since 2017 it seems that BMDP has been unavailable.

6.2.4 Testing Each Term Last

EXAMPLE 6.2.5.

Code 6.2.2.

```
rm(list = ls())
tense <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example3-7-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10a.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-7-1.dat",
  sep="",col.names=c("y","Tn","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
M=factor(Ms)
D=factor(Dr)
```

```
T=factor(Tn)
yy=log(y)


m00 = lm(yy ~ T*W*M*D)
m0a=anova(m00)
SS=m0a[,2]

tab=c(4*sqrt(SS[-16]),4*sqrt(SS[-16])/sqrt(sum(1/y)))

matrix(tab,15,2,dimnames = list(NULL,c("Est", "z")))
```

For all the weakness of using what is essentially a normal approximation for count data, because the linear model is balanced, the results in the above table do not depend on the order in which effects are fitted. R will very conveniently print out a similar ANOVA table of G^2 values (deviance reductions) and for this example the results are very similar. The only problem with the output for the code below is that the model `(T + W + M + D)^4` which is coded below gives different results than the model `(D + M + W + T)^4` or any other permutation of the factors. (Check the higher-order interactions.) Still, for these data the basic story remains pretty much the same regardless of the order. There is no guarantee that that will always happen. (Similar to tilde's, copying caret ^ from a pdf file to R may require replacing the symbol.)

Code 6.2.3.

```
rm(list = ls())
tense <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example3-7-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\tab20-10a.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-7-1.dat",
  sep="",col.names=c("y","Tn","Wt","Ms","Dr"))
attach(tense)
tense
#summary(tense)

W=factor(Wt)
M=factor(Ms)
D=factor(Dr)
T=factor(Tn)

m8 <- glm(y ~ (T + W + M + D)^4,family = poisson)
anova(m8)
```

6.3 Example of Stepwise Methods

The examples in the book involve just fitting all of the models and accumulating the results. The following subsections discuss the use of R's `step` command. You will notice that `step` applied to ANOVA type models does not eliminate redundant terms.

6.3.1 Forward Selection

The following code runs and gives results not too dissimilar from the book. It is hard for me to care enough about forward selection to worry about the differences. The main differences are due to R basing decisions on AIC values rather than other things. It can probably vary the results by changing the `k` parameter discussed in the next subsection.

Code 6.3.1.

```
rm(list = ls())
abt <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB3-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-4.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Tab3-1.dat",
  sep="",col.names=c("R","S","A","O","y"))
attach(abt)
abt
#summary(abt)

r=factor(R)
o=factor(O)
s=factor(S)
a=factor(A)

svr <- glm(y ~ r + s + o + a ,family = poisson)
 step(svr,y ~ r*s*o*a, direction="forward")
```

6.3.2 Backward Elimination

The book considers applying backward elimination to the initial model containing all two-factor terms. R's `step` command decides what to delete based on the AIC criterion rather than the P values used in the book. The default step procedure drops one less two-factor term than the procedure in the book. You can arrive at the same model that the book gets by redefining AIC. R includes a `k` parameter for AIC where

the default value (and the true definition of AIC) is `k=2`. If you reset `k=2.5`, you arrive at the same final model as the book.

Code 6.3.2.

```
rm(list = ls())
abt <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB3-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-4.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Tab3-1.dat",
  sep="",col.names=c("R","S","A","O","y"))
attach(abt)
abt
#summary(abt)

r=factor(R)
o=factor(O)
s=factor(S)
a=factor(A)

svf <- glm(y ~ (r + s + o + a)^2 ,family = poisson)
svff <- step(svf, direction="backward")
svff.stp$anova
step(svf, direction="backward",k=2.5)
step(svf, direction="backward",test="LRT")
```

The `test="LRT"` command gets G^2s printed but the process still stops based on the (possibly modified) AIC. You could perhaps adjust the `k=` option to go past your stopping point and then use the G^2 statistics to determine your stopping point.

You might find it interesting to see what the following code produces.

```
svff <- glm(y ~ r*s*o*a ,family = poisson)
svff.stp <- step(svff, direction="backward")
svff.stp$anova
```

6.4 Aitkin's Method of Backward Selection

Computationally this is just fitting a lot of models and using `pchisq` to obtain the gammas.

6.5 Model Selection Among Decomposable and Graphical Models

Computationally this is just fitting a lot of models and perhaps using `anova` to obtain differences. The trick is in selecting the models and I am not about to program that for you.

6.6 Use of Model Selection Criteria

Code 6.6.1.

```
rm(list = ls())
abt <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB3-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-4.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Tab3-1.dat",
  sep="",col.names=c("R","S","A","O","y"))
attach(abt)
abt
#summary(abt)

r=factor(R)
o=factor(O)
s=factor(S)
a=factor(A)

sv6 <- glm(y ~ r:s:o + o:a ,family = poisson)
sv5 <- glm(y ~ r:o + s:o + o:a ,family = poisson)
sv4 <- glm(y ~ r + s + o:a ,family = poisson)
sv3 <- glm(y ~ r:a + s + o:a ,family = poisson)
sv2 <- glm(y ~ r + s:o + o:a ,family = poisson)
sv1 <- glm(y ~ r:o + s + o:a ,family = poisson)
sv0 <- glm(y ~ r + s + o + a ,family = poisson)


tab6=c(6,df.residual(sv6), deviance(sv6),
deviance(sv6)-2*df.residual(sv6),
((deviance(sv0)-deviance(sv6))/(deviance(sv0))),
(1-(deviance(sv6)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv6))))  )
tab5=c(5,df.residual(sv5), deviance(sv5),
deviance(sv5)-2*df.residual(sv5),
((deviance(sv0)-deviance(sv5))/(deviance(sv0))),
```

```
(1-(deviance(sv5)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv5))))  )
tab4=c(4,df.residual(sv4), deviance(sv4),
deviance(sv4)-2*df.residual(sv4),
((deviance(sv0)-deviance(sv4))/(deviance(sv0))),
(1-(deviance(sv4)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv4))))  )
tab1=c(3,df.residual(sv1), deviance(sv1),
deviance(sv1)-2*df.residual(sv1),
((deviance(sv0)-deviance(sv1))/(deviance(sv0))),
(1-(deviance(sv1)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv1))))  )
tab2=c(2,df.residual(sv2), deviance(sv2),
deviance(sv2)-2*df.residual(sv2),
((deviance(sv0)-deviance(sv2))/(deviance(sv0))),
(1-(deviance(sv2)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv2))))  )
tab3=c(1,df.residual(sv3), deviance(sv3),
deviance(sv3)-2*df.residual(sv3),
((deviance(sv0)-deviance(sv3))/(deviance(sv0))),
(1-(deviance(sv3)*df.residual(sv0)/
(deviance(sv0)* df.residual(sv3))))  )

t(matrix(c(tab6,tab5,tab4,tab3,tab2,tab1),6,6))
```

6.7 Residuals and Influential Observations

Code 6.7.1.

```
rm(list = ls())
abt <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB3-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-4.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Tab3-1.dat",
  sep="",col.names=c("R","S","A","O","y"))
attach(abt)
abt
#summary(abt)

r=factor(R)
o=factor(O)
s=factor(S)
a=factor(A)
```

```
mm <- glm(y ~ r:s:o + o:a ,family = poisson)
mms = summary(mm)

rpearson=(y-mm$fit)/(mm$fit)^(.5)
rstand=rpearson/(1-hatvalues(mm))^(.5)
infv = c(y,mm$fit,hatvalues(mm),rpearson,rstand,
   cooks.distance(mm))
inf=matrix(infv,I(mms$df[1]+mms$df[2]),6,dimnames =
list(NULL,c("n","mhat","lev","Pearson","Stand.","C")))
inf

index=c(1:72)
plot(index,hatvalues(mm),ylab="Leverages",
   xlab="Index")
boxplot(rstand,horizontal=TRUE,
   xlab="Standardized residuals")
plot(index,rstand,ylab="Standardized residuals",
   xlab="Index")
qqnorm(rstand,ylab="Standardized residuals")
boxplot(cooks.distance(mm),horizontal=TRUE,
   xlab="Cook's distances")
plot(index,cooks.distance(mm),ylab="Cook's distances",
   xlab="Index")
```

6.8 Drawing Conclusions

Table 6.7 can be obtained with the code of Section 4.6.

6.9 Exercises

Chapter 7
Models for Factors with Quantitative Levels

7.1 Models for Two-Factor Tables

Code 7.1.1.

```
rm(list = ls())
abt <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example7-1-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\EX21-5-1.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example7-1-1.dat",
  sep="",col.names=c("c","p","y"))
attach(abt)
abt


C=factor(c)
P=factor(p)
m3 <- glm(y~C+P+C:p,family=poisson) #[C][P][C_1]
m2 <- glm(y~C+P+c:P,family=poisson) #[C][P][P_1]
m1 <- glm(y~C+P+c:p,family=poisson) #[C][P][gamma]
m0 <- glm(y~C+P,family=poisson)     #[C][P]
df=c(m3$df.residual,m2$df.residual,m1$df.residual,
     m0$df.residual)
G2=c(m3$deviance,m2$deviance,m1$deviance,m0$deviance)
A2q=G2-(2*df)
modelm=c(df,G2,A2q)
model=matrix(modelm,4,3,dimnames =
     list(NULL,c("df","G2","A-q")))
model


m1s=summary(m1)
```

```
m1s
anova(m1)

rpearson=(y-m1$fit)/(m1$fit)^(.5)
rstand=rpearson/(1-hatvalues(m1))^(.5)
infv = c(y,m1$fit,hatvalues(m1),rpearson,rstand,
   cooks.distance(m1))
inf=matrix(infv,I(m1s$df[1]+m1s$df[2]),6,dimnames =
list(NULL,c("y","yhat","lev","Pearson","Stand.","C")))
inf

m0$fit
```

7.2 Higher-Dimensional Tables

Code 7.2.1.

```
rm(list = ls())
abt <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB3-1.DAT"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\TAB21-4.DAT",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TAB3-1.DAT",
  sep="",col.names=c("R","S","A","O","y"))
attach(abt)
abt
#summary(abt)

r=factor(R)
o=factor(O)
s=factor(S)
a=factor(A)
A2=A*A
#[RSO][OA]
ab <- glm(y ~ r:s:o + o:a ,family = poisson)
abp=summary(ab)
abp
anova(ab)

#[RSO][A][O_1][O_2]
ab2 <- glm(y ~ r:s:o + a + o:A + o:A2,family = poisson)
abp2=summary(ab2)
abp2
anova(ab2)
```

```
#[RSO][A][O_1]
ab3 <- glm(y ~ r:s:o + a + o:A ,family = poisson)
abp3=summary(ab3)
abp3
anova(ab3)

rpearson=(y-ab3$fit)/(ab3$fit)^(.5)
rstand=rpearson/(1-hatvalues(ab3))^(.5)
infv = c(y,ab3$fit,hatvalues(ab3),rpearson,
  rstand,cooks.distance(ab3))
inf=matrix(infv,I(abp3$df[1]+abp3$df[2]),6,dimnames =
list(NULL,c("y", "yhat", "lev","Pearson","Stand.","C")))
inf
```

7.3 Unknown Factor Scores

Code 7.3.1.

```
rm(list = ls())
ct=c(43,16,3,6,11,10,9,18,16)
L=c(1,2,3,1,2,3,1,2,3)
V=c(1,1,1,2,2,2,3,3,3)
l=factor(L)
v=factor(V)
ind=glm(ct ~ v + l,family=poisson)
summary(ind)
t=log(ind$fit)
t2=t*t
m11=glm(ct ~ v + l + v:t,family=poisson)
summary(m11)
m12=glm(ct ~ v + l + l:t,family=poisson)
summary(m12)
m13=glm(ct ~ v + l + t2,family=poisson)
summary(m13)

summary(m11)
t
```

Also see package `logmult` which runs things from package `gnm`.

7.4 Logit Models with Unknown Scores

I must have gotten the maximum likelihood fits in Table 7.3 from Chuang (1983) because I have no idea how I would have computed them.

Code 7.4.1.

```
rm(list = ls())
High=c(245,330,388,100,77,51,28,89,102,67,87,62,
   125,234,233,109,197,90)
Low=c(115,152,153,40,37,19,11,37,35,18,12,13,68,
   91,173,47,82,32)
R=c(1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,3)
E=c(1,2,3,4,5,6,1,2,3,4,5,6,1,2,3,4,5,6)
r=factor(R)
e=factor(E)
T=cbind(High,Low)
m4=glm(T ~ r + e,family=binomial)
summary(m4)
t=log(m4$fit/(1-m4$fit))
t2=t*t
m5=glm(T ~ r + e + r:t,family=binomial)
summary(m5)
m6=glm(T ~ r + e + e:t,family=binomial)
summary(m6)
m7=glm(T ~ r + e + t2,family=binomial)
summary(m7)
```

7.5 Exercises

Chapter 8
Fixed and Random Zeros

8.1 Fixed Zeros

This just involves leaving some cells out of the table.

EXAMPLE 8.1.1. Brunswick (1971) reports data on the health concerns of teenagers.

Code 8.1.1.

```
rm(list = ls())
ct=c(4,42,57,2,7,20,9,4,19,71,7,8,10,31)
s=c(1,1,1,1,1,1,2,2,2,2,2,2,2,2)
a=c(1,1,1,2,2,2,1,1,1,1,2,2,2,2)
h=c(1,3,4,1,3,4,1,2,3,4,1,2,3,4)
S=factor(s)
A=factor(a)
H=factor(h)
m7=glm(ct ~ S:A + S:H + A:H,family=poisson)
summary(m7)
m6=glm(ct ~ S:H + A:H,family=poisson)
summary(m6)
m5=glm(ct ~ S:A + A:H,family=poisson)
summary(m5)
m4=glm(ct ~ S:A + S:H ,family=poisson)
summary(m4)
m3=glm(ct ~ S:A + H,family=poisson)
summary(m3)
m2=glm(ct ~ S:H + A,family=poisson)
summary(m2)
m1=glm(ct ~ S + A:H,family=poisson)
summary(m1)
m0=glm(ct ~ S + A + H,family=poisson)
```

```
summary(m0)


tab7=c(7,df.residual(m7),
deviance(m7),1-pchisq(deviance(m7),df.residual(m7)))
tab6=c(6,df.residual(m6),
deviance(m6),1-pchisq(deviance(m6),df.residual(m6)))
tab5=c(5,df.residual(m5),
deviance(m5),1-pchisq(deviance(m5),df.residual(m5)))
tab4=c(4,df.residual(m4),
deviance(m4),1-pchisq(deviance(m4),df.residual(m4)))
tab1=c(1,df.residual(m1),
deviance(m1),1-pchisq(deviance(m1),df.residual(m1)))
tab2=c(2,df.residual(m2),
deviance(m2),1-pchisq(deviance(m2),df.residual(m2)))
tab3=c(3,df.residual(m3),
deviance(m3),1-pchisq(deviance(m3),df.residual(m3)))
tab0=c(0,df.residual(m0),
deviance(m0),1-pchisq(deviance(m0),df.residual(m0)))

t(matrix(c(tab7,tab6,tab5,tab4,tab3,tab2,tab1,tab0),4,8))
```

8.2 Partitioning Polytomous Variables

Code 8.2.1.

```
rm(list = ls())
ct=c(104,165,65,100,4,5,13,32,42,142,44,130,3,6,6,23)
s=c(1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2)
y=c(1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2)
r=c(1,1,2,2,3,3,4,4,1,1,2,2,3,3,4,4)

S=factor(s)
Y=factor(y)
R=factor(r)

m7=glm(ct ~ R:Y + R:S + Y:S,family=poisson)
summary(m7)
m5=glm(ct ~ R:Y + Y:S,family=poisson)
summary(m5)
m4=glm(ct ~ R:Y + R:S ,family=poisson)
summary(m4)
m3=glm(ct ~ R:Y + S,family=poisson)
summary(m3)
```

```
tab7=c(7,df.residual(m7),
deviance(m7),1-pchisq(deviance(m7),df.residual(m7)))
tab5=c(5,df.residual(m5),
deviance(m5),1-pchisq(deviance(m5),df.residual(m5)))
tab4=c(4,df.residual(m4),
deviance(m4),1-pchisq(deviance(m4),df.residual(m4)))
tab3=c(3,df.residual(m3),
deviance(m3),1-pchisq(deviance(m3),df.residual(m3)))

t(matrix(c(tab7,tab4,tab5,tab3),4,4))

g=c(1,1,2,2,1,1,4,4,1,1,2,2,1,1,4,4)
h=c(1,1,1,1,3,3,1,1,1,1,1,1,3,3,1,1)
G=factor(g)
H=factor(h)
eq2=glm(ct ~ G:H:Y + G:H:S + Y:S,family=poisson)
summary(eq2)
eq3=glm(ct ~ G:H:Y + G:S + Y:S,family=poisson)
summary(eq3)

r=c(1,1,2,2,3,3,4,4,1,1,2,2,3,3,4,4)
p=c(1,1,2,2,2,2,2,2,1,1,2,2,2,2,2,2)
c=c(2,2,1,1,2,2,2,2,2,2,1,1,2,2,2,2)
j=c(2,2,2,2,1,1,2,2,2,2,2,2,1,1,2,2)
o=c(2,2,2,2,2,2,1,1,2,2,2,2,2,2,1,1)
P=factor(p)
C=factor(c)
J=factor(j)
O=factor(o)


eq4=glm(ct ~ P:C:J:O:Y + P:C:J:O:S + Y:S,family=poisson)
summary(eq4)
eq5=glm(ct ~ P:C:J:O:Y + P:S + Y:S,family=poisson)
summary(eq5)
eq6=glm(ct ~ P:C:J:O:Y + C:S + Y:S,family=poisson)
summary(eq6)
eq7=glm(ct ~ P:C:J:O:Y + J:S + Y:S,family=poisson)
summary(eq7)
eq8=glm(ct ~ P:C:J:O:Y + O:S + Y:S,family=poisson)
summary(eq8)
eq9=glm(ct ~ P:C:J:O:Y + Y:S,family=poisson)
summary(eq9)
```

```
tab4=c(4,df.residual(eq4),deviance(eq4),
  1-pchisq(deviance(eq4),df.residual(eq4)))
tab5=c(5,df.residual(eq5),deviance(eq5),
  1-pchisq(deviance(eq5),df.residual(eq5)))
tab6=c(6,df.residual(eq6),deviance(eq6),
  1-pchisq(deviance(eq6),df.residual(eq6)))
tab7=c(7,df.residual(eq7),deviance(eq7),
  1-pchisq(deviance(eq7),df.residual(eq7)))
tab8=c(8,df.residual(eq8),deviance(eq8),
  1-pchisq(deviance(eq8),df.residual(eq8)))
tab9=c(9,df.residual(eq9),deviance(eq9),
  1-pchisq(deviance(eq9),df.residual(eq9)))

t(matrix(c(tab4,tab5,tab6,tab7,tab8,tab9),4,6))
```

8.3 Random Zeros

EXERCISE 8.1. Modify the code of Example 2.5.3 to include the SWP.

EXAMPLE 8.3.1.

In the book it indicates that 3 rows of the 24×3 table are zeros as can be seen in Table 8.3 of the book. If you just fit the entire data, the corresponding 9 fitted values $\hat{m}_{hijk}$ are converging to 0. The way the data are read below, those cases turn out to be 9, 11, 19, 33, 35, 43, 57, 59, 67. *It reports the same G^2 as in the book, but does not give the book's degrees of freedom.*

Code 8.3.1.

```
rm(list = ls())
knee <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB8-3.DAT"),
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TAB8-3.DAT",
  sep="",col.names=c("hh","ii","jj","kk","ct"))
attach(knee)
knee
#summary(knee)
T=factor(hh)
S=factor(ii)
A=factor(jj)
R=factor(kk)

art=glm(ct ~ T:A:S + T:R + A:R,family=poisson)
```

```
summary(art)
art$fit


art=glm(ct ~ T:A:S + T:A:R,family=poisson)
summary(art)
```

Note also the much larger than usual number of iterations the computations take. Technically, there are no maximum likelihood estimates because some estimates are converging to 0 and 0 is not an allowable MLE.

Now we drop the offending cells, refit the models, and get the same G^2 values but the degrees of freedom from the book.

Code 8.3.2.

```
ctt=ct
ctt[9]=NA
ctt[11]=NA
ctt[19]=NA
ctt[33]=NA
ctt[35]=NA
ctt[43]=NA
ctt[57]=NA
ctt[59]=NA
ctt[67]=NA

artt=glm(ctt ~ T:A:S + T:R + A:R,family=poisson)
summary(artt)


artt=glm(ctt ~ T:A:S + T:A:R,family=poisson)
summary(artt)
```

EXAMPLE 8.3.2.

In the book it indicates that 3 rows of the 24×3 table are zeros as can be seen in Table 8.4 of the book. If you just fit the entire data, the 12 cases identified in the book have fitted values $\hat{m}_{hijk}$ are converging to 0. The way the data are read below, those cases turn out to be 3, 4, 12, 13, 14, 15, 19, 21, 28, 30, 31, 34. Also as indicated in the book, this is a saturated model, so the other cases have $\hat{m}_{hijk} = n_{hijk}$. The program reports $G^2 \doteq 0$ on 4 degrees of freedom which, again, is too many degrees of freedom

Code 8.3.3.

```
rm(list = ls())
mel <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB8-4.DAT"),
```

```
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TAB8-4.DAT",
  sep="",col.names=c("hh","ii","jj","kk","ct"))
attach(mel)
mel
#summary(mel)
G=factor(hh)
R=factor(ii)
Im=factor(jj)
S=factor(kk)

out=glm(ct~G:R:Im+G:R:S+G:Im:S+R:Im:S,family=poisson)
summary(out)
out$fit
ct
```

Again note the much larger than usual number of iterations the computations take. Technically, there are no maximum likelihood estimates because some estimates are converging to 0 and 0 is not an allowable MLE.

Now we drop the offending cells and refit the model, we get the same $G^2 \doteq 0$ value but 0 degrees of freedom as in the book.

Code 8.3.4.

```
ctt=ct
ctt[3]=NA
ctt[4]=NA
ctt[12]=NA
ctt[13]=NA
ctt[14]=NA
ctt[15]=NA
ctt[19]=NA
ctt[21]=NA
ctt[28]=NA
ctt[30]=NA
ctt[31]=NA
ctt[34]=NA

out=glm(ctt~G:R:Im+G:R:S+G:Im:S+R:Im:S,family=poisson)
summary(out)
out$fit
ct
```

8.4 Exercises

It would seem that at some point I intended to give some computational advice on these exercises.

EXAMPLE 8.4.3. *Partitioning Two-Way Tables*. Lancaster (1949) and Irwin (1949)

EXAMPLE 8.4.4. *The Bradley-Terry Model*.

Chapter 9
Generalized Linear Models

No computing in this chapter.

9.1 Distributions for Generalized Linear Models

9.2 Estimation of Linear Parameters

9.3 Estimation of Dispersion and Model Fitting

9.4 Summary and Discussion

9.5 Exercises

Generalizations of Poisson log-linear models with additional variability can be obtained from two forms of negative binomial regression. Both have mean m but have variance either (NB1) $\phi m \equiv m + \alpha m$ or (NB2) $m + \theta m^2$. NB1 can be fitted in `glm` with `family = quasipoisson`. NB2 involves a modified form of `glm`, `glm.nb`. See https://www.rdocumentation.org/packages/MASS/versions/7.3-57/topics/glm.nb

Chapter 10
The Matrix Approach to Log-Linear Models

10.1 Maximum Likelihood Theory for Multinomial Sampling

10.2 Asymptotic Results

EXAMPLE 10.2.3. In the abortion opinion data of Table 3.1 and the model [RSO][OA] (cf. book Table 6.7 generated by the code in Sectiong 4.6), examine the cell for nonwhite males between 18 and 25 years of age who support abortion.

EXAMPLE 10.2.4. *Automobile Injuries*

Code 10.2.1.

```
rm(list = ls())
sb <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example3-2-4.dat"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\EX21-3-1.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-2-4.dat",
  sep="",col.names=c("nuLL","i","j","k","y"))

attach(sb)
#summary(sb)
I=factor(i)
J=factor(j)
K=factor(k)
m7 <- glm(y ~ I:J + I:K + J:K,family = poisson)
m7s=summary(m7)

rpearson=(y-m7$fit)/(m7$fit)^(.5)
rstand=rpearson/(1-hatvalues(m7))^(.5)
infv = c(y,m7$fit,hatvalues(m7),rpearson,rstand,
```

```
    cooks.distance(m7))
inf=matrix(infv,I(m7s$df[1]+m7s$df[2]),6,dimnames =
list(NULL,c("y","yhat","lev","Pearson","Stand.","C")))
inf
```

EXAMPLE 10.2.6. *Classroom behavior.*

Code 10.2.2.

```
rm(list = ls())
sb <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/Example3-2-4.dat"),
#"C:\\E-drive\\Books\\ANREG2\\newdata\\EX21-3-2.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\Example3-2-4.dat",
  sep="",col.names=c("y","i","j","k"))

attach(sb)
#summary(sb)
I=factor(i)
J=factor(j)
K=factor(k)
m7 <- glm(y ~ I + J+ K+ J:K,family = poisson)
m7s=summary(m7)
vcov(m7)

rpearson=(y-m7$fit)/(m7$fit)^(.5)
rstand=rpearson/(1-hatvalues(m7))^(.5)
infv = c(y,m7$fit,hatvalues(m7),rpearson,rstand,
     cooks.distance(m7))
inf=matrix(infv,I(m7s$df[1]+m7s$df[2]),6,dimnames =
list(NULL,c("y","yhat","lev","Pearson","Stand.","C")))
inf
```

10.3 Product-Multinomial Sampling

10.4 Inference for Model Parameters

10.5 Methods for Finding Maximum Likelihood Estimates

See Section 3.3.

10.6 Regression Analysis of Categorical Data

EXAMPLE 10.6.1. *Drug Comparisons.*

Code 10.6.1.

```
rm(list = ls())
cnt=c(6,16,2,4,2,4,6,6)
a=c(1,1,1,1,2,2,2,2)
A=3-2*a
b=c(1,1,2,2,1,1,2,2)
B=3-2*b
c=c(1,2,1,2,1,2,1,2)
AB=A*B
C=3-2*c
y=log(cnt)


ts <- lm(y ~ A+B+AB+C,weights = cnt)
tsp=summary(ts)
tsp
anova(ts)

# new standard errors
coef(tsp)[,2]/tsp$sigma
# new z scores
coef(tsp)[,3]*tsp$sigma
```

10.7 Residual Analysis and Outliers

10.8 Exercises

Chapter 11
The Matrix Approach to Logit Models

There is no computing in this chapter.

11.1 Estimation and Testing for Logistic Models

11.2 Model Selection Criteria for Logistic Regression

11.3 Likelihood Equations and Newton-Raphson

11.4 Weighted Least Squares for Logit Models

11.5 Multinomial Response Models

11.6 Asymptotic Results

11.7 Discrimination, Allocations, and Retrospective Data

11.8 Exercises

Chapter 12
Maximum Likelihood Theory for Log-Linear Models

There is no computing in this chapter.

12.1 Notation

12.2 Fixed Sample Size Properties

12.3 Asymptotic Properties

12.4 Applications

12.5 Proofs of Lemma 12.3.2 and Theorem 12.3.8

Chapter 13
Bayesian Binomial Regression

The book concerns itself with using a discrete approximation to the posterior distribution: $\Pr[\beta = \beta^r|Y] = \tilde{q}_r$, $r = 1,\ldots,t$. Only in Section 4 does it discuss importance sampling as a method for obtaining a discrete approximation. (Ed Bedrick did almost all the Bayesian computing in the second edition for joint papers we wrote with Wes Johnson.) Here we focus on *Markov chain Monte Carlo (McMC)* methods and use the entire chapter to gradually introduce the computations needed. McMC (ideally) provides a sample from the posterior distribution so the discrete approximation is taken as $\Pr[\beta = \beta^r|Y] = 1/t$, $r = 1,\ldots,t$. Most of the computations are in R and are based on saved samples from the posterior distribution. *Such samples can be obtained from a number of packages.* We focus on *OpenBUGs* but also illustrate *JAGS*. (*JAGS* seems to be a better option for Apple users.) Another viable option is *STAN*. In an extra section at the end of the chapter we discuss the benefits of using the OpenBUGS GUI to interactively determine whether a Markov chain has converged so that it provides samples from the posterior. SAS's `proc genmod` also has an option for fitting Bayesian models.

Bayesian computation has made huge strides since the second edition of this book in 1997. McMC methods provide a sequence of simulated observations on the parameters. Two particular tools in this approach are Gibbs sampling (named by Geman and Geman, 1984, after distributions that were introduced by, and named after, the greatest American physicist of the 19th century) and the Metropolis-Hastings algorithm (named after the the alphabetically listed authors of Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller, 1953, and the person who introduced the technique to the statistics community [and made a simple but very useful improvement] Hastings, 1970). For more about McMC see Christensen et al. (2010, Chapter 6), the references therein, the references in the text, and also Tierney (1994) gives a good brief presentation the the underlying theory. Henceforth Christensen et al. is referred to as *BIDA*.

Unlike importance samples, McMC samples are not naturally independent and they only become (approximate) samples from the posterior distribution after the Markov chain has been running quite a while, i.e., after one is deep into the sequence of observations. Computationally, we need to specify a *burn-in* period for

the samples to get close to the posterior and then we throw away all of the observations from the burn-in period. Among the samples we use, we tend to take larger samples sizes to adjust for their lack of independence. When averaging (functions of) sample observations to estimate some quantity (including probabilities of events) the lack of independence is rarely a problem (due to the Ergodic theorem). When applying more sophisticated techniques than averaging to the samples, if independence is important to the technique, we can often approximate independence by *thinning* the sample, i.e., using, say, only every 10th or 20th observation from the Markov chain. Of course in any program we need to specify the sample size and the rate at which any thinning occurs. The default is typically no thinning.

BUGS (Bayesian inference Using Gibbs Sampling) provides a language for specifying Bayesian models computationally, cf. http://www.mrc-bsu.cam.ac.uk/software/bugs/. OpenBUGS (https://www.mrc-bsu.cam.ac.uk/software/bugs/openbugs/) and JAGS (http://mcmc-jags.sourceforge.net/) implement that language to actually analyze data. We illustrate running BUGS code using OpenBUGS through R via R2OpenBUGS. An alternative package for this is BRugs. The use of JAGS will be illustrated in R with R2jags. JAGS has somewhat different input and output commands from OpenBUGS. A more recent BUGS oriented platform is NIMBLE. Also, Plummer (2023) reviews Bayesian simulations methods.

Although we present results using OpenBUGS, its newest version is called MultiBUGS, which is designed to increase speed using parallel processing. MultiBUGS should have an identical interface to OpenBUGS. An outdated version of OpenBUGS is WinBUGS, cf. https://cran.r-project.org/web/packages/R2WinBUGS/index.html. WinBUGS was used in *BIDA* but the commands given in *BIDA* should almost all work with OpenBUGS. (Some data entry is different.) STAN is an alternative McMC program but does not use the BUGS programming language for specifying models. STAN will not be illustrated but can also be run in R through RStan. R also has tools for doing McMCs directly, cf. https://cran.r-project.org/web/packages/MCMCpack/index.html. Again, *most of the code given here does not depend on how you obtained your posterior samples.*

My goal is to get you through an McMC version of the computations in the book. For a general tutorial on BUGS see http://www.openbugs.net/Manuals/Tutorial.html or the book by Lunn et al. (2013). Fletcher Christensen provided code to illustrate computations performed in *BIDA* on the O-ring and Trauma data. His code has been substantially modified and expanded. Penny Darsey made substantial contributions and Davis Dotson got JAGS running for me.

13.1 Introduction

There are three things you need to do in using OpenBUGS or JAGS:

- Specify the Bayesian model. This involves specifying both the sampling distribution and the prior distribution.
- Enter the data. Depending on how you specified the model, this includes specifying any "parameters" in the model that are known.
- Identify and give starting values for the unknown parameters.

There are a lot of similarities between the R and BUGS languages but before proceeding we mention a couple oddities of the BUGS language. First,

$$y \sim \text{Bin}(N, p)$$

is written as

```
y ~ dbin(p,N)
```

with the order of N and p reversed. Also,

$$y \sim N(m, v)$$

is written as

```
y ~ dnorm(m,1/v)
```

where the variance v is replaced in BUGS by the *precision*, $1/v$. Replacing normal variances with precisions is a very common thing to do in Bayesian analysis because it simplifies many computations.

The main thing is to specify the model inside a statement: `model{}`. For simple problems this is done by specifying a sampling distribution and a prior distribution. In Subsection 13.2.1 of the book we mentioned that the standard approach has been to use either a normal distribution for β or the improper "noninformative" diffuse prior $\pi(\beta) = 1$, so we begin using those.

The sampling model for the O-ring data with failures y_i and temperatures τ_i is

$$y_i \sim \text{Bin}(1, p_i),$$

$$\text{logit}(p_i) \equiv \log\left(\frac{p_i}{1-p_i}\right) = \beta_1 + \beta_2 \tau_i, \quad i = 1, \ldots, 23.$$

For programming convenience we have relabeled the intercept as β_1 and the slope as β_2. In the BUGS language this can be specified as

```
for(i in 1:23){
  y[i] ~ dbin(p[i],1)
  logit(p[i]) <- beta[1] + beta[2]*tau[i]
  }
```

The parameters of primary interest are `beta[1]` and `beta[2]`. We also need to specify initial values for the unknown parameters but that is not part of specifying the model. Remember if you copy the tilde symbol ~ from a .pdf file into a program like R or OpenBUGS, you may need to delete and replace the symbol.

The standard approach has been to specify normal priors or flat priors. For example, we might specify independent priors

$$\beta_j \sim N(a_j, 1/b_j), \quad j = 1,2,$$

for some specified values, say, $a_1 = 10$, $b_1 = 0.001$, $a_2 = 0$, $b_2 = 0.004$. The b_js are precisions (inverse variances). I have heretically put almost no thought into this prior other than making the precisions small. In BUGS the prior model is most directly specified as

```
beta[1] ~ dnorm(10,.001)
beta[2] ~ dnorm(0,.004)
```

In my opinion one should always explore different priors to examine how sensitive the end results are to the choice of prior. It is also my opinion that one of those priors should be your best attempt to quantify your prior knowledge about the unknown parameters.

All together the model is

```
model{
 for(i in 1:23){
   y[i] ~ dbin(p[i],1)
   logit(p[i]) <- beta[1] + beta[2]*tau[i]
   }
   beta[1] ~ dnorm(10,.001)
   beta[2] ~ dnorm(0,.004)
}
```

The model is only a part of an overall program for OpenBUGS or JAGS. To run BUGS models within R, we need to place this model into a .txt file, say, `Oring.txt`.

More generally we could use a multivariate normal prior on β by using R to prespecify a mean vector and a precision matrix (the inverse of the covariance matrix). In our example,

```
a = c(10,0)
B=matrix(c(.001,0,0,.004),ncol=2)
```

and then write the model as

```
model{
 for(i in 1:23){
   y[i] ~ dbin(p[i],1)
   logit(p[i]) <- beta[1] + beta[2]*tau[i]
   }
   beta[1:2] ~ dmnorm(a[],B[,])
}
```

The improper "noninformative" diffuse prior $\pi(\beta) = 1$ mentioned in Subsection 13.2.1 can be specified in the model as

```
model{
 for(i in 1:23){
   y[i] ~ dbin(p[i],1)
   logit(p[i]) <- beta[1] + beta[2]*tau[i]
   }
   beta[1] ~ dflat()
   beta[2] ~ dflat()
}
```

Producing a functional program involves identifying the model, specifying the data, and specifying initial values for all of the unknown parameters. We have two choices on how to run this. We can run it through R or we can run it through the OpenBUGS GUI. We begin by running it through R because that requires a more transparent specification of the various steps involved. The GUI is more flexible and is discussed in the last section of this chapter. I recommend you learn to use it before doing serious data analysis.

What follows is an R program for an analysis of the O-ring data. The various parts are explained using comments within the program.

Code 13.1.1.

```
  # Clear previous work
rm(list = ls())
  # Enter O-ring Data
y=c(1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0)
tau=c(53,57,58,63,66,67,67,67,68,69,70,70,70,70,72,
73,75,75,76,76,78,79,81)

# Call the R2OpenBUGS library.
# Set paths so that R2OpenBUGS can find and store
# the files it needs, including the model file Oring.txt
# and where you have stored OpenBUGS on your computer.
library(R2OpenBUGS)
BUGS_path <-
"c:\\Program Files (x86)\\OpenBUGS\\OpenBUGS323\\OpenBUGS.exe"
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
myworking_dir <- "c:\\E-drive\\Books\\LOGLIN3\\BAYES\\"

# Define the McMC sample characteristics.
# This code gives (we hope) 10,000 nonindependent
# observations from the posterior distribution.
iterates <- 10000
burn_in <- 1000

# Identify the data
data <- list( "y", "tau")
```

```
# Identify the parameters
parameters <- list("beta")

# Identify parameter initial values in a list of lists.
inits <- list(list(beta=c(0,0)))
# The program is designed to run more than one McMC chain.
# This is a list of lists to allow different initial
# values for different chains.

# Easier than the list of lists is to generate
# initial values via a function.
inits <- function () { list(beta = c(0,0)) }
# It would be redundant to run both inits <- commands

# Putting all the pieces together:
Oring <- bugs( data, inits, parameters,
         model.file=
         "c:\\E-drive\\Books\\LOGLIN3\\BAYES\\Oring.txt",
         n.chains=1, n.iter=iterates+burn_in,
         n.thin=1, n.burnin=burn_in,
         OpenBUGS.pgm=BUGS_path,
         working.directory=myworking_dir,
         debug=F )
Oring$summary
# The following command tells you what info
# is contained in Oring
summary(Oring)
# For example
Oring$mean
Oring$sd
Oring$median
```

Rather than defining `y` and `tau` separately you could also write the `data` statement as

```
data <- list(
y=c(1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0),
tau=c(53,57,58,63,66,67,67,67,68,69,70,70,70,70,72,
73,75,75,76,76,78,79,81)
)
```

If you change the last command in `bugs` to `debug=T`, R will actually open an OpenBUGS GUI and you can manipulate the process manually.

13.1.1 Alternative Specifications

Sometimes it is more convenient to define the prior more obliquely. Rather than the direct specification we used, we can specify

```
for(j in 1:2){ beta[j] ~ dnorm(a[j],b[j]) }
```

where at this point in the code it is not clear whether the `a[j]`s and `b[j]`s are unknown parameters or not. In either case we have to specify values for them in a `list` statement, say,

```
list(a=c(10,0), b=c(.001,.004))
```

Including these list entries as part of the `data` list, rather than as initial values, completes our prior specification (and implicitly defines them as not being parameters).

13.2 Bayesian Inference: O-ring Data

To this point we have introduced Bayesian computation for logistic regression but we have not yet reproduced any results from the book. We now do that and also produce the new graphics used in the third edition.

Whereas the book goes back and forth between the O-ring and Trauma data sets, for computational simplicity, *in this section we only deal with the O-ring data.* The next section, tentatively marked as 13.22, will deal with the Trauma data. Section 3 (immediately after Section 22), deals with Section 3 of the book, diagnostics.

13.2.1 Specifying the Prior and Approximating the Posterior

Section 1 illustrates Bayesian computations using a normal prior. The main addition in this section is that we demonstrate how to program the induced prior on the regression coefficients as discussed in the book. This involves modifications to the BUGS model. But we also fill out many details of the analysis.

13.2.1.1 Constructing Figures 13.1 through 13.4

Figures 13.1 through 13.4 are constructed analytically and do not involve any simulations from the posterior distribution. Future subsubsections provide estimated versions of these plots derived from OpenBUGS simulations, so we show all of the plots for your comparison. The estimated versions are much less smooth.

The bivariate densities need to be evaluated on a grid. Visually, we only care about the shape of the bivariate densities so we rescale them to suit our convenience.

FIGURE 13.1 (PRIOR). We evaluate the density on a $B \times B$ grid of points and construct a contour plot.

Code 13.2.1.

```
rm(list = ls())
# plot the density function pi()
B=1000  #size of square grid
# initialize pi as a B by B matrix of 1s
pi=rep(1,B*B)
pi=matrix(pi,B,B)

# Grid domain
beta1 = seq(-15,25,length=B)
beta2 = seq(-.4,.2,length=B)

# evaluate prior density on grid
rk=2  #rank of X
eta=rep(1,B*rk)
eta=matrix(eta,B,rk)
tautilde=c(55,75)
ytilde=c(1,.577)
Ntilde=c(1.577,1.577)

for(j in 1:B)
{
for(k in 1:rk)
{
eta[,k] = beta1 + beta2[j]*(tautilde[k])
pi[,j] = pi[,j] * ((exp(eta[,k])/(1+exp(eta[,k])))**ytilde[k])*
((1-(exp(eta[,k])/(1+exp(eta[,k]))))**(Ntilde[k]-ytilde[k]))
}
}

# rescale because we only care about
# shape of density
pi=pi*1000 #/(sum(pi)/(60*.8))

contour(beta1,beta2,pi,nlevels=8,
ylab=expression(beta["1"]),xlab=expression(beta[0]))
```

FIGURE 13.2 (POSTERIOR). Pretty much the same as the previous plot except computing the kernel of the posterior is more involved since it involves contributions from both the prior density and the likelihood function.

Code 13.2.2.

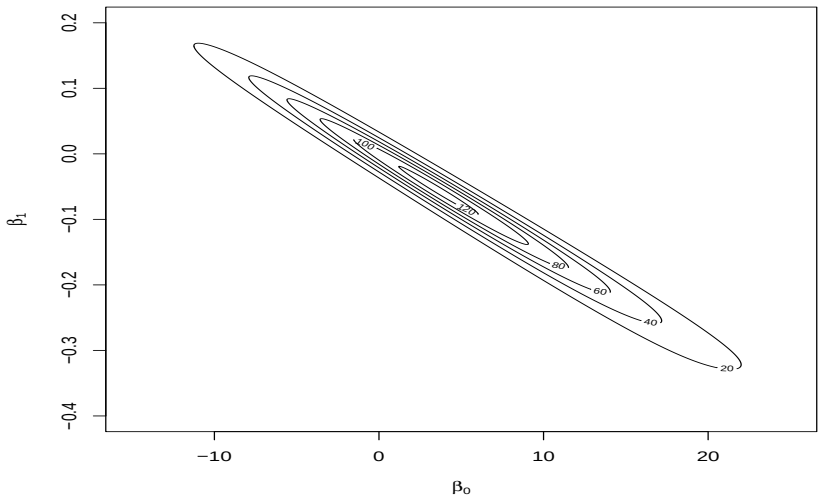


Fig. 13.1 Book Figure 13.1; O-Ring Data: Contour shapes of prior on β.

```
rm(list = ls())
# plot density pi
B=1000  #size of square grid
#pi a B by B matrix of 1s
pi=rep(1,B*B)
pi=matrix(pi,B,B)

# Grid domain --- changed from previous plot
beta1 = seq(0,25,length=B)
beta2 = seq(-.4,.0,length=B)

# contribution of prior density
rk=2  #rank of X
eta=rep(1,B*rk)
eta=matrix(eta,B,rk)
tautilde=c(55,75)
ytilde=c(1,.577)
Ntilde=c(1.577,1.577)

for(j in 1:B)
{
for(k in 1:rk)
{
```

```
eta[,k] = beta1 + beta2[j]*(tautilde[k])
pi[,j] = pi[,j] * ((exp(eta[,k])/(1+exp(eta[,k])))**ytilde[k])*
((1-(exp(eta[,k])/(1+exp(eta[,k]))))**(Ntilde[k]-ytilde[k]))
}
}

# contribution of likelihood
y=c(1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0)
N=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
tau=c(53,57,58,63,66,67,67,67,68,69,70,70,70,70,72,
73,75,75,76,76,78,79,81)
mt=mean(tau)
n=length(y)

etaD=rep(1,B*n)
etaD=matrix(etaD,B,n)
for(j in 1:B)
{
for(k in 1:n)
{
etaD[,k] = beta1 + beta2[j]*(tau[k])
pi[,j] = pi[,j] * ((exp(etaD[,k])/(1+exp(etaD[,k])))**y[k])*
((1-(exp(etaD[,k])/(1+exp(etaD[,k]))))**(N[k]-y[k]))
}
}

pi=pi*10000000 #/(sum(pi)/(60*.8))

contour(beta1,beta2,pi,nlevels=8,
ylab=expression(beta["1"]),xlab=expression(beta[0]))
```

FIGURE 13.3 (MEAN ADJUSTED PRIOR). The number 69.565 is the mean of the 23 τ_i values. Instead of fitting $\text{logit}(p_i) = \beta_1 + \beta_2\tau_i$ we are fitting $\text{logit}(p_i) = \beta_1 + \beta_2(\tau_i - 69.565)$. Otherwise, pretty much the same as Fig. 13.1

Code 13.2.3.

```
rm(list = ls())
# plot density pi
B=1000  #size of square grid
#pi a B by B matrix of 1s
pi=rep(1,B*B)
pi=matrix(pi,B,B)

# Grid domain
beta1 = seq(-5,5,length=B)
```

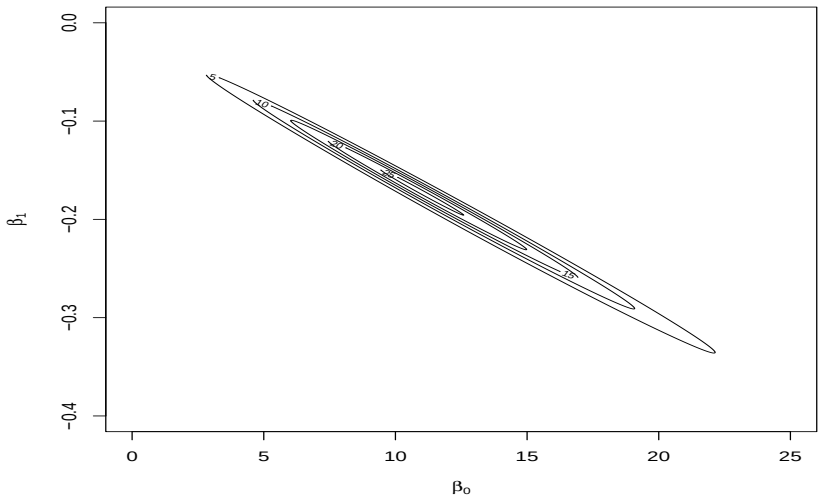


Fig. 13.2 Book Figure 13.2; O-Ring Data: Contour shapes of posterior on β.

```
beta2 = seq(-.5,.3,length=B)

#prior density
rk=2  #rank of X
eta=rep(1,B*rk)
eta=matrix(eta,B,rk)
tautilde=c(55,75)
ytilde=c(1,.577)
Ntilde=c(1.577,1.577)


for(j in 1:B)
{
for(k in 1:rk)
{
eta[,k] = beta1 + beta2[j]*(tautilde[k]-69.565)
pi[,j] = pi[,j] * ((exp(eta[,k])/(1+exp(eta[,k])))**ytilde[k])*
((1-(exp(eta[,k])/(1+exp(eta[,k]))))**(Ntilde[k]-ytilde[k]))
}
}

pi=pi*1000#/(sum(pi)/(60*.8))
```

```
contour(beta1,beta2,pi,nlevels=8,
ylab=expression(beta["1"]),xlab=expression(beta[0]))
```

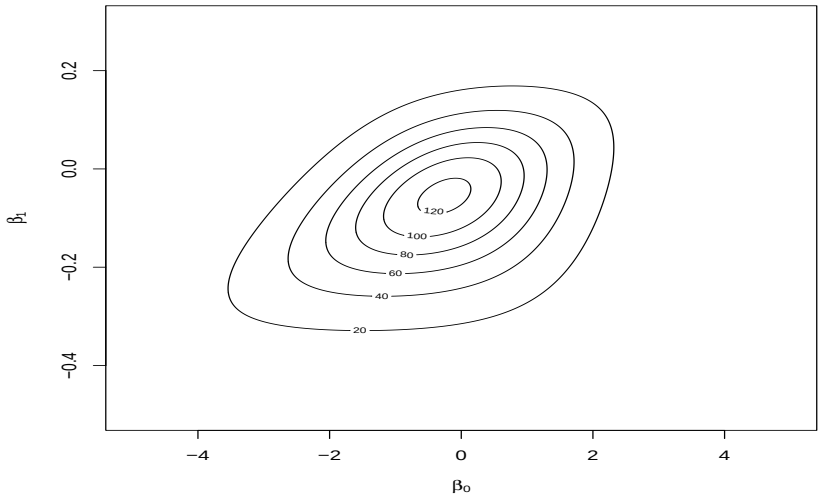


Fig. 13.3 Book Figure 13.3; O-Ring Data: Contour shapes of centered prior on β.

FIGURE 13.4 (MEAN CORRECTED POSTERIOR). Pretty much the same as Figure 13.2 except mean corrected like Figure 13.3.

Code 13.2.4.

```
rm(list = ls())
# plot density pi
B=1000  #size of square grid
#pi a B by B matrix of 1s
pi=rep(1,B*B)
pi=matrix(pi,B,B)

# Grid domain
beta1 = seq(-3,1,length=B)
beta2 = seq(-.4,.0,length=B)

#prior density
rk=2  #rank of X
eta=rep(1,B*rk)
eta=matrix(eta,B,rk)
```

```
tautilde=c(55,75)
ytilde=c(1,.577)
Ntilde=c(1.577,1.577)

for(j in 1:B)
{
for(k in 1:rk)
{
eta[,k] = beta1 + beta2[j]*(tautilde[k]-69.565)
pi[,j] = pi[,j] * ((exp(eta[,k])/(1+exp(eta[,k])))**ytilde[k])*
((1-(exp(eta[,k])/(1+exp(eta[,k]))))**(Ntilde[k]-ytilde[k]))
}
}

y=c(1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0)
N=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
tau=c(53,57,58,63,66,67,67,67,68,69,70,70,70,70,72,
73,75,75,76,76,78,79,81)
mt=mean(tau)
n=length(y)

etaD=rep(1,B*n)
etaD=matrix(etaD,B,n)
for(j in 1:B)
{
for(k in 1:n)
{
etaD[,k] = beta1 + beta2[j]*(tau[k]-69.565)
pi[,j] = pi[,j] * ((exp(etaD[,k])/(1+exp(etaD[,k])))**y[k])*
((1-(exp(etaD[,k])/(1+exp(etaD[,k]))))**(N[k]-y[k]))
}
}

pi=pi*10000000#/(sum(pi)/(60*.8))

contour(beta1,beta2,pi,nlevels=8,
ylab=expression(beta["1"]),xlab=expression(beta[0]))
```

Instead of doing direct computations from the prior and posterior, we can sample β and use the samples to estimate the prior and posterior. Figures 13.5 and 13.6 are based on such estimates. The computations for them are given later.

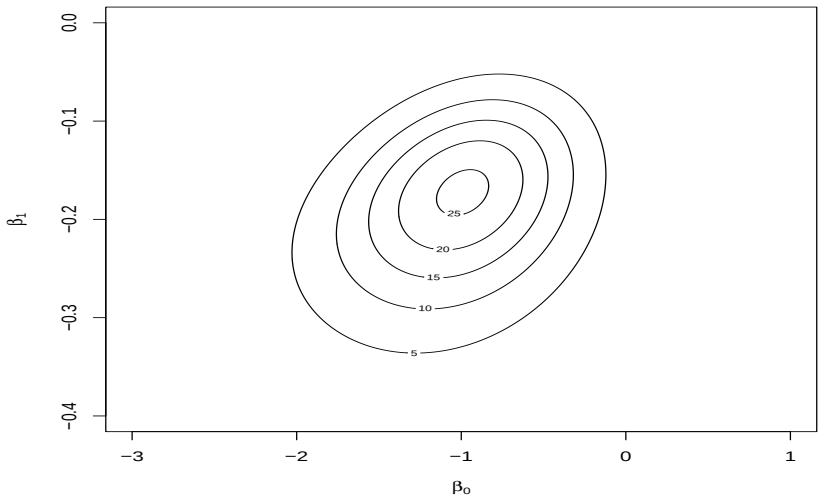


Fig. 13.4 Book Figure 13.4; O-Ring Data: Contour shapes of centered posterior on β.

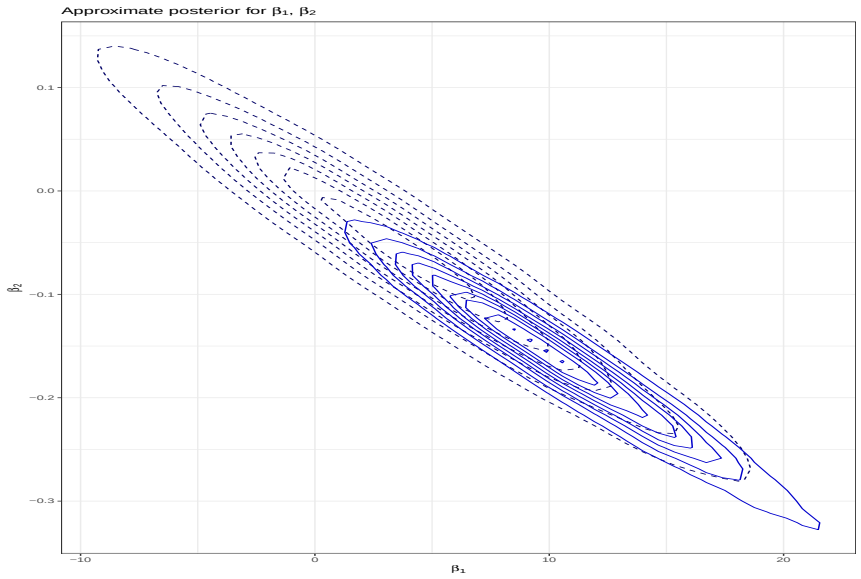


Fig. 13.5 Book Figures 13.1 and 13.2; O-Ring Data: Estimated contours of prior and posterior on β.

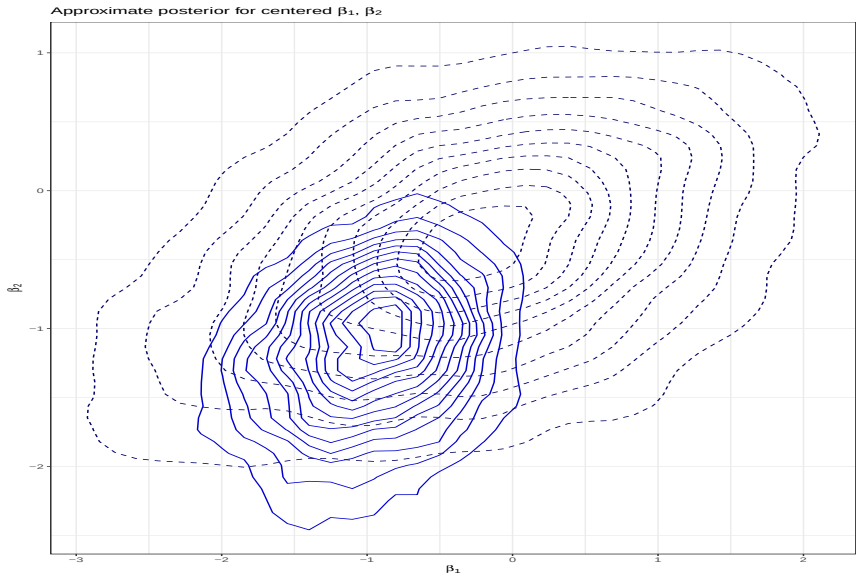


Fig. 13.6 Book Figures 13.3 and 13.4; O-Ring Data: Estimated contours of centered prior and posterior on β.

13.2.1.2 Induced Prior

The book elicits independent priors on the probabilities $\tilde{p}_1$ and $\tilde{p}_2$ of O-ring failure associated with the temperatures $\tilde{x}_1 \equiv \tilde{\tau}_1 = 55$ and $\tilde{x}_2 \equiv \tilde{\tau}_2 = 75$. It then induces a prior on the regression coefficients from the elicited prior. The elicited prior consists of independent Beta distributions,

$$\tilde{p}_j \sim \text{Beta}(a_j, b_j), \quad j = 1,2$$

$$(a_1, a_2) = (1, 0.577) \quad (b_1, b_2) = (0.577, 1)$$

Sampling from the induced prior on β is actually easier than determining the prior analytically. For the logistic model,

$$\begin{bmatrix} \text{logit}(\tilde{p}_1) \\ \text{logit}(\tilde{p}_2) \end{bmatrix} = \begin{bmatrix} 1 & 55 \\ 1 & 75 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix},$$

so

$$\begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} 1 & 55 \\ 1 & 75 \end{bmatrix}^{-1} \begin{bmatrix} \text{logit}(\tilde{p}_1) \\ \text{logit}(\tilde{p}_2) \end{bmatrix} = \begin{bmatrix} 75/20 & -55/20 \\ -1/20 & 1/20 \end{bmatrix} \begin{bmatrix} \text{logit}(\tilde{p}_1) \\ \text{logit}(\tilde{p}_2) \end{bmatrix}.$$

The equality allows us to transform samples from the $\tilde{p}_j$s directly into samples of the β_js. In BUGS this is easily programmed as

```
# Elicited Prior
for(j in 1:2){ ptilde[j] ~ dbeta(a[j],b[j]) }
 # Induced prior on the regression coefficients
 beta[1] <- (75/20)*logit(ptilde[1])
            - (55/20)*logit(ptilde[2])
 beta[2] <-  (-1/20)*logit(ptilde[1])
            +  (1/20)*logit(ptilde[2])
```

The values `a[1], a[2], b[1], b[2]` all need to be specified as part of a **data** `list` statement. (Note that the prior used in this book is different than the similar prior used in *BIDA*.) This code is a large part of our model which is specified in the next subsubsection.

In general we define an invertible square matrix of predictor variables $\tilde{X}$ and elicit distributions for a vector of probabilities $\tilde{p}$ associated with the rows of $\tilde{X}$. The probabilities are related to the β coefficients via

$$F(\tilde{X}\beta) = \tilde{p} \quad \text{or} \quad \beta = \tilde{X}^{-1} F^{-1}(\tilde{p}).$$

In logistic regression, F^{-1} is just the function that maps each probability to its logit. For models with an intercept and more than one predictor variable (like the Trauma data), we will want to invert $\tilde{X}$ in R, but in this code we have inverted the matrix $\tilde{X}$ analytically, since it is only 2×2. Samples from the distribution of $\tilde{p}$ are easily transformed into samples of the regression coefficients.

13.2.1.3 BUGS Preliminaries

As inputs to an overall R program Fletch produced a .csv data file and two .txt model files, one for the uncentered and one for centered data. Both incorporate the induced prior information but with the actual parameters of the beta distributions yet to be specified. (These would work for completely different priors as long as the priors were specified at 55 and 75 degrees.)

The model for the **uncentered** data is saved as `O-ring_model_a.txt` and consists of

```
model{
for( i in 1:n ){
y[i] ~ dbin( p[i], 1)
logit( p[i] ) <- beta[1] + beta[2] * temp[i]
}
for( j in 1:2 ){
ptilde[j] ~ dbeta( a[j], b[j] )
}
beta[1] <- (75/20) * logit( ptilde[1] )
               - (55/20) * logit( ptilde[2] )
beta[2] <- (-1/20) * logit( ptilde[1] )
               + (1/20) * logit( ptilde[2] )
}
```

Actually, you can even create the model file in R with the commands

```
Oring_model_a<-cat("model{
  for( i in 1:n ){
  y[i]  ~ dbin( p[i], 1)
  logit( p[i] ) <- beta[1] + beta[2] * temp[i]
  }
  for( j in 1:2 ){
  ptilde[j]  ~ dbeta( a[j], b[j] )
  }
  beta[1] <- (75/20) * logit( ptilde[1] )
                - (55/20) * logit( ptilde[2] )
  beta[2] <- (-1/20) * logit( ptilde[1] )
                + (1/20) * logit( ptilde[2] )}",
                       file="Oring_model_a.txt")
```

The model for the **centered data** was saved as `O-ring_model_b.txt` With the centered data, the invertible square matrix $\tilde{X}$ is more complicated. In particular,

$$\tilde{X} = \begin{bmatrix} 1 & 55-69.56522 \\ 1 & 75-69.56522 \end{bmatrix} = \begin{bmatrix} 1 & -14.56522 \\ 1 & 5.43478 \end{bmatrix}, \quad \tilde{X}^{-1} = \begin{bmatrix} 0.27174 & 0.72826 \\ -0.05000 & 0.05000 \end{bmatrix}.$$

The centered model involves the variable

```
c.temp[i]
```

which is essentially

```
temp[i]-69.56522
```

It will need to be defined in the R program before running OpenBUGS. The centered model becomes,

```
model{
for( i in 1:n ){
y[i] ~ dbin( p[i], 1)
logit( p[i] ) <- beta[1] + beta[2] * c.temp[i]
}
for( j in 1:2 ){
ptilde[j] ~ dbeta( a[j], b[j] )
}
beta[1] <- 0.2717391 * logit( ptilde[1] )
              + 0.7282609 * logit( ptilde[2] )
beta[2] <- -0.05 * logit( ptilde[1] )
              + 0.05 * logit( ptilde[2] )
}
```

The **data file**, which is ordered by Flight Number rather than Temperature (cf., book Table 2.1), is called `O-ring_data.csv` and looks like

```
"Temperature","Failure"
66,0
70,1
69,0
68,0
67,0
72,0
73,0
70,0
57,1
63,1
70,1
78,0
67,0
53,1
67,0
75,0
70,0
81,0
76,0
79,0
75,1
```

```
76,0
58,1
```

Unlike the book, Fletch has not ordered the temperatures from smallest to largest. The book draws special attention to case 18 which is case 21 in Fletch's data file. Thus, to delete case 18, I will incorporate a command `y[21]=NA`.

13.2.1.4 Generate and Save Posterior samples: OpenBUGS

This subsubsection is the only place in this section that we will use the `bugs` *program from the* `R2OpenBUGS` *package.* Everything else is simply R programming.

Fletch wrote one large program but I have broken it into more compact independent pieces. The first piece generates and saves the β^r samples, $r = 1,\ldots,t$. In the code t is denoted `interates`. These saved iterates will be read back into later pieces of code.

Uncentered Data: We need to run this 3 times with minor variations to store posterior samples from three conditions: the original code, a larger McMC sample that is heavily thinned, and the original code with case 18 deleted.

We begin by using `bugs` to produce the posterior samples β^r as row vectors concatenating them into a $t \times k$ matrix with $k = 2$. We transpose this into a $k \times t$ matrix before saving them.

Code 13.2.5.

```
#######################
##
##  O-Ring Analysis
##
rm(list = ls())
library(R2OpenBUGS)


BUGS_path <-
"c:\\Program Files (x86)\\OpenBUGS\\OpenBUGS323\\OpenBUGS.exe"
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
working_dir <- "c:\\E-drive\\Books\\LOGLIN3\\BAYES\\"
model_a_filename <- "O-ring_model_a.txt"
model_b_filename <- "O-ring_model_b.txt"


iterates <- 10000 # "t" from book, later =5000
burn_in <- 1000

########################
```

```
##  O-ring data

ORing_data <-
read.csv(url("http://stat.unm.edu/~fletcher/LLM/DATA/O-ring_data.csv"))

n <- dim(ORing_data)[1]
y <- ORing_data$Failure
temp <- ORing_data$Temperature

# Delete case 18 in book
# which is case/flight 21 in Fletch's file
#y[21]=NA


### Fletch used BIDA not LOGLIN Priors
#   so I have changed them
#a <- c( 1.6, 1 )
#b <- c( 1, 1.6 )
# LOGLIN priors
a <- c( 1, .577 )
b <- c( .577, 1 )


data <- list( "n", "y", "temp", "a", "b" )

inits <- function() {
  list( ptilde = c( 0.5, 0.5 ) )
}

parameters <- list( "beta" )

ORing.sim <- bugs( data, inits, parameters,
                   model.file=model_a_filename,
                   n.chains=1, n.iter=iterates+burn_in,
                   n.thin=1, n.burnin=burn_in,
                   OpenBUGS.pgm=BUGS_path,
                   working.directory=working_dir,
                   debug=F)
ORing.sim$summary
beta <- t(ORing.sim$sims.list$beta)
# save 2 x t matrix of posterior beta iterates for later use
save(beta,file="post-samp.Rda")
# save case 18 deleted beta
# save(beta,file="post-samp-18.Rda")
# save thinned samples of beta
```

```
# save(beta,file="post-samp-thin.Rda")
```

Now `beta` is the $2 \times t$ matrix of posterior samples with columns β^r. Again, $t =$ `interates`. In subsequent programs we will use `load("post-samp.Rda")` to return `beta` for us. We will also need to run this code uncommenting `y[21]=NA` and save the posterior samples as `post-samp-18.Rda`. Finally, for producing Figure 13.10, we will want to rerun this code (for all the data) with `interates=5000` and within the `bugs` command `n.thin=100` and save the results in `post-samp-thin.Rda`. (The number of iterates are those reported after the thinning has occurred.)

Centered Data:

To analyze the centered model run this modified program. At the very end of the program there are lines to comment and uncomment for running the data without case 18 and saving the results.

Code 13.2.6.

```
rm(list = ls())
library(R2OpenBUGS)


BUGS_path <-
"c:\\Program Files (x86)\\OpenBUGS\\OpenBUGS323\\OpenBUGS.exe"
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
working_dir <- "c:\\E-drive\\Books\\LOGLIN3\\BAYES\\"
model_a_filename <- "O-ring_model_a.txt"
model_b_filename <- "O-ring_model_b.txt"


iterates <- 10000
burn_in <- 1000

########################
##  O-ring data

ORing_data <- read.csv("O-ring_data.csv", header=T)

n <- dim(ORing_data)[1]
y <- ORing_data$Failure
# To rerun the analysis without obs. 18
# uncomment the line following the next
# case 18 in book is case 21 in Fletch's data
# y[21]=NA
temp <- ORing_data$Temperature
centered_temp <- scale(temp, scale=F)
```

```
c.temp <- centered_temp[,1]
temp
c.temp

### Fletch used BIDA not LOGLIN3 Priors
#   so I have changed them
#a <- c( 1.6, 1 )
#b <- c( 1, 1.6 )
# LOGLIN3 priors
a <- c( 1, .577 )
b <- c( .577, 1 )


data <- list( "n", "y", "c.temp", "a", "b" )

inits <- function() {
  list( ptilde = c( 0.5, 0.5 ) )
}

parameters <- list( "beta" )


ORing.c.sim <- bugs( data, inits, parameters,
                     model.file=model_b_filename,
                     n.chains=1, n.iter=iterates+burn_in,
                     n.thin=1, n.burnin=burn_in,
                     OpenBUGS.pgm=BUGS_path,
                     working.directory=working_dir,
                     debug=F)
# save 2 x t matrix of posterior iterates
beta.c <- t(ORing.c.sim$sims.list$beta)
save(beta.c,file="post-c-samp.Rda")
#save(beta.c,file="post-c-samp-18.Rda")
```

These posterior samples are used for constructing estimated versions of the bivariate posterior density plots for β. We will also use the centered results to construct plots of predictive distributions. These prediction plots could have been produced from the uncentered results but Fletch used the centered results.

13.2.1.5 Generate and Save Posterior samples: JAGS

We run the uncentered results in JAGS, the commands are very similar to OpenBUGS but you employ `jags` rather than `bugs`. Thanks to Davis Dotson for working on this. Again, the posterior samples β^r are saved in the $2 \times t$ matrix `beta`.

Code 13.2.7.

```
rm(list = ls())
library(R2jags)

setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
working_dir <- "c:\\E-drive\\Books\\LOGLIN3\\BAYES\\"
ORing_data <- read.csv("O-ring_data.csv", header=T)

iterates <- 10000
burn_in <- 1000

n <- dim(ORing_data)[1]
y <- ORing_data$Failure
temp <- ORing_data$Temperature
# The next two commands are only needed
# for running the centered model
centered_temp <- scale(temp, scale=F)
c.temp <- centered_temp[,1]

a <- c( 1, .577 )
b <- c( .577, 1 )
inits <- function() {
  list( ptilde = c( 0.5, 0.5 ) )
}

sim.dat.jags<-list("y","n","a","b","temp")
ORing.params<-c("beta")
ORing.fit <- jags(data = sim.dat.jags, inits = inits,
  parameters.to.save = ORing.params, n.chains = 1, n.thin=1,
n.iter = iterates+burn_in, n.burnin = burn_in,
model.file = "O-ring_model_a.txt")
beta <- t(ORing.fit$BUGSoutput$sims.list$beta)
#save(beta,file="post-samp.Rda")
summary(ORing.fit$BUGSoutput$sims.list$beta)
```

To run the centered data, within the `jags` command, just switch to

```
model.file = "O-ring_model_b.txt"
```

and in the penultimate line, change the save command to

```
save(beta,file="post-c-samp.Rda")
```

13.2.1.6 Estimated Bivariate Density Plots

Fletch wrote a program constructing estimated versions of Figures 13.1 through 13.4. Fletch used the plotting package `ggplot` which is very popular but unfamiliar to me.

Uncentered data

Fletch's program produces estimated versions of Figures 13.1 and 13.2 in a single plot. Figures 13.1 and 13.2 display the prior and posterior (uncentered) distributions analytically, but Fletch sampled the prior and saved both the prior samples and the posterior samples into one data frame. First we reload the posterior sample matrix `beta`. Then we create the prior samples and a data frame that contains both samples for uncentered data.

To this point we have freely used the `logit` function in defining BUGS models. Another difference between BUGS and R is that R does not have a predefined logit function, so we need to define one.

Code 13.2.8.

```
rm(list = ls())
# Set working directory and read posterior samples
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("post-samp.Rda")
# respecify number of iterates
iterates=length(beta[1,])
# respecify prior parameters
a <- c( 1, .577 )
b <- c( .577, 1 )
# define logit function
logit <- function(p)
{
lg = log(p/(1-p))
return(lg)}
# Sample from prior ptildes and transform to betas
prior_p1 <- rbeta(iterates, a[1], b[1])
prior_p2 <- rbeta(iterates, a[2], b[2])
prior_beta1 <-
   (75/20) * logit( prior_p1 ) - (55/20) * logit( prior_p2 )
prior_beta2 <-
   (1/20) * logit( prior_p2 ) - (1/20) * logit( prior_p1 )

# Save prior and posterior samples
# for later use in a data frame.
beta_frame <- data.frame( cbind( prior_beta1,
                     prior_beta2, beta[1,], beta[2,] ) )
```

```
names( beta_frame ) <-
  c( "prior_beta1", "prior_beta2",
           "posterior_beta1", "posterior_beta2" )
save(beta_frame,file="beta-samples.Rda")
```

Although the posterior samples in `beta` were previously saved in `post-samp.Rda`, Fletch's new data frame `beta_frame`, saved in `beta-samples.Rda`, has given them a new name to distinguish them from them prior samples that are also in the data frame. Once we obtain Fletch's estimated figures, we will revert to using our saved version of `beta`.

We begin the plotting program by reloading the prior and posterior samples of β.

Code 13.2.9.

```
rm(list = ls())
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("beta-samples.Rda")
library(ggplot2)
# library(boot)  #this was part of Fletch's program
# but I don't see it being used anywhere
# "boot" is a bootstrapping library
ggplot( beta_frame ) +
  geom_density2d( aes(x=prior_beta1, y=prior_beta2),
  color="#000066", linetype=2 ) +
  geom_density2d( aes(x=posterior_beta1, y=posterior_beta2),
  color="#0000cc" ) +
  theme_bw() +
  xlab( expression( beta[0] ) ) +
  ylab( expression( beta[1] ) ) +
  ggtitle( expression(
  paste( "Approximate prior and posterior for ",
  beta[0], ", ", beta[1] ) ) )
```

Centered data

This is Fletch's code to obtain an estimated version of Figures 13.3 and 13.4 in a single plot.

We begin by reloading the posterior samples from the centered data and creating the prior samples. We then create a data frame that contains both for centered data. The primary change is that centering changes the second column of $\tilde{X}$ and thus changes $\tilde{X}^{-1}$.

Code 13.2.10.

```
rm(list = ls())
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("post-c-samp.Rda")
```

```
# When running without case 18
# comment previous "load" and uncomment next line
# load("post-c-samp-18.Rda")
iterates=length(beta.c[1,])
a <- c( 1, .577 )
b <- c( .577, 1 )
# define logit function
logit <- function(p)
{
lg = log(p/(1-p))
return(lg)}

prior_p1 <- rbeta(iterates, a[1], b[1])
prior_p2 <- rbeta(iterates, a[2], b[2])
prior_beta_c_1 <- 0.2717391 * logit( prior_p1 ) +
                    0.7282609 * logit( prior_p2 )
prior_beta_c_2 <- -0.05 * logit( prior_p1 ) +
                          0.05 * logit( prior_p2 )

beta_c_frame <- data.frame( cbind( prior_beta_c_1,
        prior_beta_c_2, beta.c[1,], beta.c[2,] ) )

# When running without case 18
# comment out next 2 commands
names( beta_c_frame ) <- c( "prior_beta_c_1",
"prior_beta_c_2","posterior_beta_c_1", "posterior_beta_c_2" )
save(beta_c_frame,file="beta-c-samp.Rda")
# When running without case 18 uncomment next 2 commands (4 lines)
#names( beta_c_frame ) <- c( "prior_beta_c_18_1",
#"prior_beta_c_18_2","posterior_beta_c_18_1",
# "posterior_beta_c_18_2" )
#save(beta_c_frame,file="beta-c-samp-18.Rda")
```

Reloading the centered prior and posterior samples, Fletch again uses `ggplot`.

Code 13.2.11.

```
rm(list = ls())
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("beta-c-samp.Rda")
library(ggplot2)
ggplot( beta_c_frame ) +
  geom_density2d( aes(x=prior_beta_c_1, y=prior_beta_c_2),
  color="#000066", linetype=2 ) +
  geom_density2d( aes(x=posterior_beta_c_1,
  y=posterior_beta_c_2),
  color="#0000cc" ) +
```

```
theme_bw() +
xlab( expression( beta[0] ) ) +
ylab( expression( beta[1] ) ) +
ggtitle( expression( paste(
"Approximate prior and posterior for centered ",
beta[0], ", ", beta[1] ) ) )
```

The program involves doing density estimation from random samples of the prior. Just to warn you, I have two slightly different versions of this plot and Penny Darsey also got a different version.

13.2.2 Predictive Probabilities

We use the posterior β samples to generate samples of predictive probabilities at different temperatures. These allow us to produce Figures 13.7 and 13.8. Figure 13.7 involves the MLEs from Chapter 2 and the Bayesian results both with and without case 18. The credible intervals of Figure 13.8 require finding percentiles of the discrete approximation to the posterior of $F(x'\beta)$. Using McMC, finding percentiles of the discrete approximation simplifies to finding percentiles of the posterior sample $F(x'\beta^r)$. In Section 13.5 we illustrate a somewhat different approach in which we program the predictive probabilities into the BUGS model rather then compute them from the saved posterior samples of the model parameters.

Using the previously generated posterior samples β^r, $r = 1,\ldots,t$, to find predictive probabilities at locations x_{fi}, $i = 1,\ldots,n_f$, we compute $F(x'_{fi}\beta^r)$, $r = 1,\ldots,t$, $i = 1,\ldots,n_f$ and average over r. Here F is the cumulative distribution function that defines our binomial sampling model.

To be more succinct, create matrices

$$X_f \equiv \begin{bmatrix} x'_{f1} \\ \vdots \\ x'_{fn_f} \end{bmatrix} \qquad \text{and} \qquad B \equiv [\beta^1, \quad \cdots, \quad \beta^t\,].$$

We need to compute

$$F(X_f B),$$

where the one-dimensional cdf F is being applied to each number in the matrix $X_f B$. We then compute the row means of this $n_f \times t$ matrix to compute the predictive probabilities and find percentiles of these samples to find the credible interval. In the code below we have B identified as `beta`.

I have greatly modified Fletch's program for constructing Figure 13.8 and also produce Figure 13.7. Fletch used the centered Bayesian model for predictions but I incorporated uncentered MLEs. Prediction results should be equivalent for the centered and uncentered models.

Code 13.2.12.

```
rm(list = ls())
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("post-c-samp.Rda")
# beta.c

# Range of predictions
prediction_temps <-  (300:800)/10

# Center prediction times
tau=c(53,57,58,63,66,67,67,67,68,69,70,70,70,70,72,
73,75,75,76,76,78,79,81)
temp_center=prediction_temps - mean(tau)
# Prediction Model Matrix
# Create a vector of 1s
J=1+temp_center - temp_center
# Create matrix X_f
Xf=c(J,temp_center)
Xf=matrix(Xf,ncol=2)
Xf

# compute posterior mean predictions
Pr=Xf %*% beta.c
Pr=exp(Pr)/(1+exp(Pr))
MP=rowMeans(Pr)
# look at prediction probabilities
matrix(c(prediction_temps,MP),ncol=2)

# posterior predictive probabilities:
#         5%, 95% quantiles and median
prediction_quantiles <- apply(Pr, 1,
                     quantile, probs=c(0.05,0.5,0.95))

# Now prepared to do Fig 13.8 because it is
# all based on the basic sample of 10000 betas
# Easy to replace posterior means
# (MP) with posterior medians (Median)
# Fig 13.8, Fletch
predictions <- data.frame(
    prediction_temps, t(prediction_quantiles), MP )
names(predictions) <-
     c("Temp", "Lower", "Median", "Upper","MP")
library(ggplot2)
ggplot( predictions ) +
  geom_line( aes(x=Temp, y=Lower),
```

```
    color="#aa0000", linetype=2 ) +
    geom_line( aes(x=Temp, y=MP),
    color="#aa0000", linetype=1 ) +
    geom_line( aes(x=Temp, y=Upper),
    color="#aa0000", linetype=2 ) +
    coord_cartesian( ylim=c(0,1) ) +
    theme_bw() +
    xlab("Temperature") +
    ylab("Failure Probability") +
    ggtitle( expression( paste(
    "O-Ring failure probability by temperature" ) ) )

  # Additional computations for Fig 13.7

  # Case 18 deleted, mean predictions
  load("post-c-samp-18.Rda")
  Pr=Xf %*% beta.c
  Pr=exp(Pr)/(1+exp(Pr))
  MP_18=rowMeans(Pr)

  # MLE predictions from Chap 2
  beta1mle=15.0429
  beta2mle=-.2322
  #uncentered MLEs
  lp= beta1mle + prediction_temps*beta2mle
  mle_predictions <- exp(lp) / ( 1 + exp(lp) )

  # Fig 13.7
  plot(prediction_temps,mle_predictions,type="l",
       xlim=c(30,85),
       ylab="Probability",xlab="Temperature",lty=3)
  lines(prediction_temps,MP,type="l",lty=1)
  lines(prediction_temps,MP_18,type="l",lty=2)
  legend("bottomleft",
       c("MLE","Bayes: full data","Bayes: no case 18"),
       lty=c(3,1,2))
```

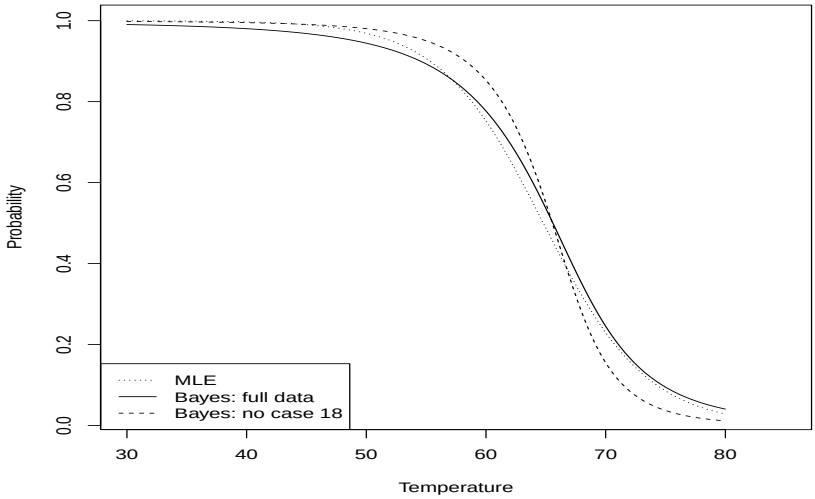


Fig. 13.7 O-Ring Data: Predictive Probabilities and MLE Predictions

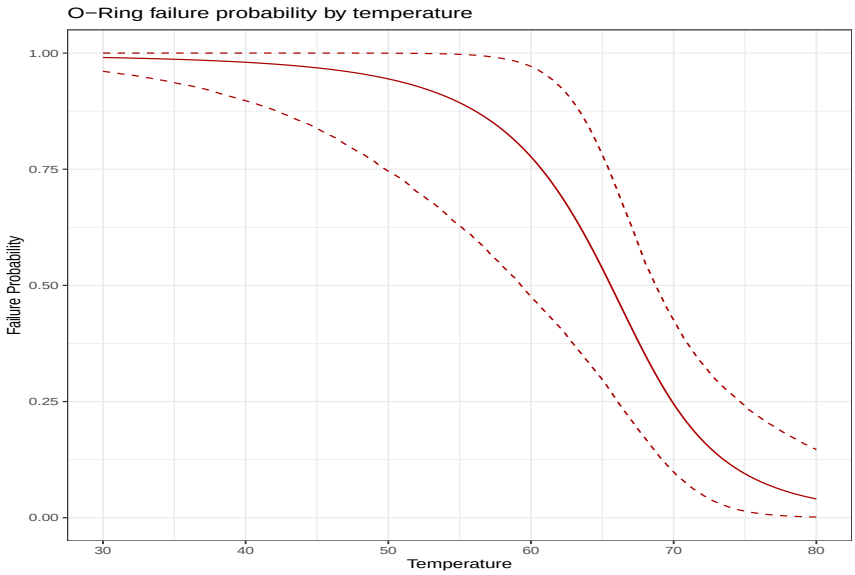


Fig. 13.8 O-Ring Data: Predictive Probabilities and 90% Intervals

13.2.3 Inference for Regression Coefficients

This involves merely summarizing our posterior samples. We used the uncentered version.

Table 13.1 Posterior Marginal Distribution from importance sampling: O-Rings

	Full Data		Case 18 Deleted	
	β_0	β_1	β_0	β_1
$\hat{\beta}_i = \mathrm{E}(\beta_i \mid Y)$	12.97	−0.2018	16.92	−0.2648
Std. Dev.$(\beta_i \mid Y)$	5.75	0.0847	7.09	0.1056
5%	4.56	−0.355	6.85	−0.459
25%	9.04	−0.251	11.98	−0.324
50%	12.44	−0.194	16.13	−0.252
75%	16.20	−0.144	20.86	−0.191
95%	23.38	−0.077	29.96	−0.114

Code 13.2.13.

```
rm(list = ls())
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
```

```
load("post-samp.Rda")
rowMeans(beta)
apply(beta, 1, mean) # Redundant command
sdv=c(sd(beta[1,]),sd(beta[2,]))
sdv
#apply(beta, 1, sd) # Redundant command
apply(beta, 1, quantile,
    probs=c(0.025,.25,0.5,.75,0.975))
```

Repeat on `post-samp-18.Rda` and `post-samp-thin.Rda`. To reproduce the book modify Subsubsection 13.2.1.4 to create another file with $t = 1000000$ `interates` and with 10,000 iterates change the prior to the *BIDA* prior as indicated in the code.

To get Figure 13.10 as similar as possible to the second edition I reran the basic code but reduced the iterates to `interates=5000` while imposing severe thinning with `n.thin=100`. I saved the new posterior iterates in the data file `post-samp-thin.Rda`, so we begin by reading them back in.

Code 13.2.14.

```
rm(list = ls())
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("post-samp-thin.Rda")
post_slope=density(beta[2,])
plot(post_slope,main="")
```

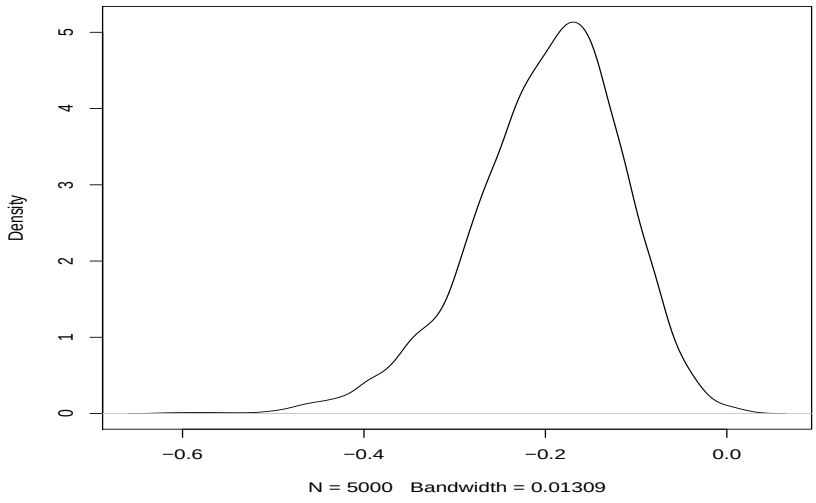


Fig. 13.10 O-Ring Data: (Estimated) Marginal Posterior Density for β_1 (slope not intercept).

13.2.4 Inference for LD_α

We begin by reading in the previously saved posterior samples β^r from `post-samp.Rda` as the $2 \times t$ matrix `beta`. We need to repeat the process with posterior samples obtained when deleting case 18.

Code 13.2.15.

```
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("post-samp.Rda")
# load("post-samp-18.Rda")
#create a placeholder matrix for the output
out=rep(1,15)
out=matrix(out,ncol=3)

af=c(.9,.75,.5,.25,.1)
for(k in 1:5){
lda=(log(af[k]/(1-af[k]))-beta[1,])/beta[2,]
out[k,]=quantile(lda,c(.05,.5,.95))
}
out
```

Table 13.2 Posterior Summaries for LD_α's

	Full Data Percentiles				Case 18 Deleted Percentiles		
α	5%	50%	95%	α	5%	50%	95%
0.90	30.2	52.9	60.4	0.90	39.8	55.1	61.2
0.75	43.4	58.5	64.0	0.75	48.9	59.4	64.0
0.50	55.9	64.2	68.5	0.50	57.5	63.8	67.5
0.25	65.1	69.8	76.4	0.25	64.1	68.1	73.0
0.10	70.3	75.4	88.3	0.10	68.3	72.4	80.9

13.22 Bayesian Inference: Trauma Data

Whereas the book goes back and forth between the O-ring and Trauma data sets, for computational simplicity, *in this section we only deal with the Trauma data.* It is tentatively marked as 13.22. Section 3 follows and deals with Section 3 of the book: Diagnostics.

The trauma data file is coded with 1 as survived. The book treats 1 as death. `TRAUMAa.DAT` is a version of `TRAUMA.DAT` without the column names.

We begin by creating the box plots of book Figure 13.5.

Code 13.22.1.

```
rm(list = ls())
Trauma <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TRAUMAa.DAT"),
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TRAUMAa.DAT",
  sep="",col.names=c("ID","Death","ISS","TI","RTS","AGE"))
# Alternative file read
#"url("http://stat.unm.edu/~fletcher/LLM/DATA/TRAUMA.DAT"),
#   sep="",header=T)
attach(Trauma)
summary(Trauma)
# From the summary, 0.93 percent "die" in the ER
# It is labeled backwards.
Death=1-Death

par(mfrow=c(2,2))
boxplot(ISS~Death,ylab="ISS")
boxplot(RTS~Death)
boxplot(AGE~Death)
par(mfrow=c(1,1))
```

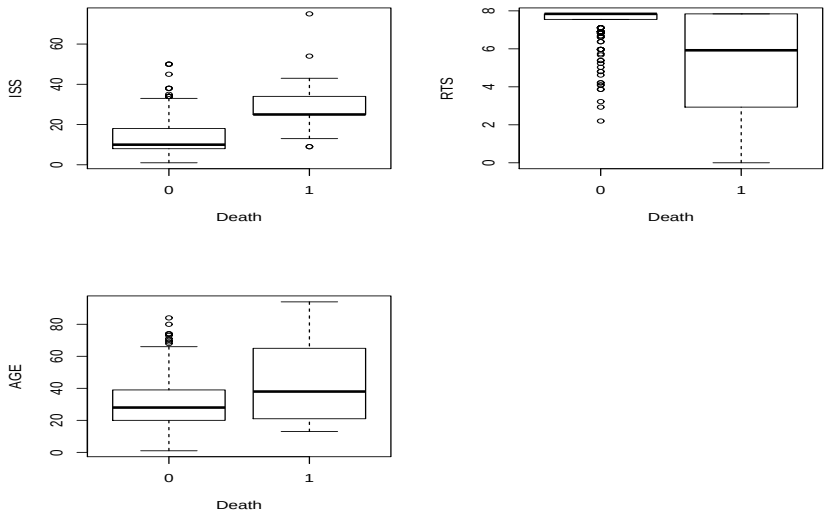


Fig. 13.11 Trauma Data: Box Plots, 1 indicates death.

13.22.1 Specifying the Prior and Approximating the Posterior

13.22.1.1 Creating the BUGS model files

The model with flat priors on the regression coefficients is pretty easy. Save this as `trauma_flat.txt`.

```
model{
 for(i in 1:n){
   Death[i] ~ dbern(p[i])
   logit(p[i]) <- beta[1]
                  + beta[2]*ISS[i] + beta[3]*RTS[i]
                  + beta[4]*AGE[i] + beta[5]*TI[i]
                  + beta[6]*AGE[i]*TI[i]
 }
 for(i in 1:6){ beta[i] ~ dflat() }
}
```

It takes more work to define the model with the informative prior. The informative prior is determined by Table 13.3. In particular, we need to deal with the $\tilde{X}$ matrix and find its inverse. We will use R to compute $\tilde{X}^{-1}$, called `Xtinv`, and include it as data for the BUGS model.

Table 13.3 Trauma Data: Prior Specification

	Design for Prior	Beta $(\tilde{y}_i, \tilde{N}_i - \tilde{y}_i)$	
i	$\tilde{x}_i'$	$\tilde{y}_i$	$\tilde{N}_i - \tilde{y}_i$
1	1 25 7.84 60 0 0	1.1	8.5
2	1 25 3.34 10 0 0	3.0	11.0
3	1 41 3.34 60 1 60	5.9	1.7
4	1 41 7.84 10 1 10	1.3	12.0
5	1 33 5.74 35 0 0	1.1	4.9
6	1 33 5.74 35 1 35	1.5	5.5

Save the following model as `trauma_model.txt` The BUGS command `inprod(x,y)` computes $x'y$.

```
model{
 for(i in 1:n){
   Death[i] ~ dbern(p[i])
   logit(p[i]) <- beta[1] + beta[2]*ISS[i]
                  + beta[3]*RTS[i]
                  + beta[4]*AGE[i]
                  + beta[5]*TI[i]
                  + beta[6]*AGE[i]*TI[i]
 }
```

```
 for(i in 1:6){
   ptilde[i] ~ dbeta(a[i],b[i])
   v[i] <- log(ptilde[i]/(1-ptilde[i])) }
 for(i in 1:6){
   beta[i] <- inprod(Xtinv[i,1:6], v[1:6])
 }
}
```

13.22.1.2 Generate and Save Posterior samples

This subsubsection is the only place in this section that we will use the `bugs` *program from the* `R2OpenBUGS` *package.* Everything else is simply R programming.

Informative prior

Code 13.22.2.

```
rm(list = ls())
# enter hyperparameters of priors
# and data size
n=300
a=c(1.1,3,5.9,1.3,1.1,1.5)
b=c(8.5,11,1.7,12,4.9,5.5)
iterates=80000
burn_in=5000
library(R2OpenBUGS)
BUGS_path <-
"c:\\Program Files (x86)\\OpenBUGS\\OpenBUGS323\\OpenBUGS.exe"
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
working_dir <- "c:\\E-drive\\Books\\LOGLIN3\\BAYES\\"

# Enter data
Trauma <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TRAUMAa.DAT"),
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TRAUMAa.DAT",
  sep="",col.names=c("ID","Death","ISS","TI","RTS","AGE"))
attach(Trauma)
Death=1-Death

#Enter Xtilde
Xtp=c(
 1, 25, 7.84, 60, 0,  0,
 1, 25, 3.34, 10, 0,  0,
 1, 41, 3.34, 60, 1, 60,
 1, 41, 7.84, 10, 1, 10,
```

```
 1, 33, 5.74, 35, 0,  0,
 1, 33, 5.74, 35, 1, 35)
Xtilde=t(matrix(Xtp,6,6))
Xtilde

# Find inverse of Xtilde
Xtinv=solve(Xtilde)
Xtinv

data <- list( "n", "Death", "a", "b" ,"Xtinv","ISS","RTS","AGE","TI")

inits <- function() {
  list(ptilde=c(0.5,0.5,0.5,0.5,0.5,0.5))
}

parameters <- list( "beta" )


Trauma.sim <- bugs( data, inits, parameters,
                   model.file="trauma_model.txt",
                   n.chains=1, n.iter=iterates+burn_in,
                   n.thin=1, n.burnin=burn_in,
                   OpenBUGS.pgm=BUGS_path,
                   working.directory=working_dir,
                   debug=F )
Trauma.sim$summary
# save p x t matrix of gamma posterior iterates
beta <- t(Trauma.sim$sims.list$beta)
save(beta,file="trauma_samp.Rda")
#save(beta,file="trauma-samp-thin.Rda")
```

Results of code with 80,000 samples, 5000 burnin.

```
                 mean         sd      2.5%        25%
beta[1]   -1.85438137 1.11013140  -4.06900  -2.587000
beta[2]    0.06594937 0.02103582   0.02677   0.051320
beta[3]   -0.59752132 0.14364672  -0.89260  -0.691725
beta[4]    0.04811565 0.01410091   0.02165   0.038550
beta[5]    1.10883893 1.08537427  -1.05500   0.404875
beta[6]   -0.01696064 0.02783206  -0.07249  -0.035500
deviance 105.28460500 2.82950050 101.30000 103.300000
              50%        75%        97.5%
beta[1]   -1.82600  -1.092000   0.30610000
beta[2]    0.06562   0.079660   0.10990000
beta[3]   -0.59380  -0.498300  -0.32779750
beta[4]    0.04761   0.057360   0.07641025
```

```
beta[5]    1.11600   1.820000   3.24200000
beta[6]   -0.01642   0.001655   0.03716050
deviance 104.80000 106.800000 112.20000000
```

Results of code with only 8,000 samples, 5000 burnin, but thining of 100.

```
                mean         sd         2.5%         25%
beta[1]   -1.78115435 1.13648017  -4.06107500  -2.5172500
beta[2]    0.06540472 0.02127947   0.02396875   0.0509575
beta[3]   -0.59949944 0.14455717  -0.89090250  -0.6944000
beta[4]    0.04736977 0.01374953   0.02196975   0.0380600
beta[5]    1.09652375 1.11537470  -1.12207500   0.3635250
beta[6]   -0.01714907 0.02834145  -0.07347225  -0.0365325
deviance 105.37631250 2.82022240 101.30000000 103.3000000
                50%         75%         97.5%
beta[1]    -1.783500 -1.01100000   0.42111750
beta[2]     0.065290  0.07952250   0.10720250
beta[3]    -0.596900 -0.50167500  -0.31839750
beta[4]     0.046925  0.05643250   0.07528025
beta[5]     1.099500  1.85000000   3.21200000
beta[6]    -0.016445  0.00236125   0.03776025
deviance 1104.900000 06.90000000 112.00000000
```

To obtain results with the **flat prior** we change the model file within the `bugs` program and we change the `inits` command and delete extraneous material including extraneous `data` inputs. (OpenBUGS likes you to use *all* the data you input to it.)

Code 13.22.3.

```
rm(list = ls())
# enter data size
n=300
iterates=80000
burn_in=5000
library(R2OpenBUGS)
BUGS_path <-
"c:\\Program Files (x86)\\OpenBUGS\\OpenBUGS323\\OpenBUGS.exe"
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
working_dir <- "c:\\E-drive\\Books\\LOGLIN3\\BAYES\\"

# Enter data
Trauma <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TRAUMAa.DAT"),
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TRAUMAa.DAT",
  sep="",col.names=c("ID","Death","ISS","TI","RTS","AGE"))
attach(Trauma)
Death=1-Death
```

```
data <- list( "n", "Death","ISS","RTS","AGE","TI")

inits <- function() {
  list(beta=c(0.5,0.5,0.5,0.5,0.5,0.5))
}

parameters <- list( "beta" )


Trauma.sim <- bugs( data, inits, parameters,
                  model.file="trauma_flat.txt",
                   n.chains=1, n.iter=iterates+burn_in,
                   n.thin=1, n.burnin=burn_in,
                   OpenBUGS.pgm=BUGS_path,
                   working.directory=working_dir,
                   debug=F )
Trauma.sim$summary
# save p x t matrix of gamma posterior iterates
beta <- t(Trauma.sim$sims.list$beta)
save(beta,file="trauma_flat_samp.Rda")
```

results of code

```
                mean         sd         2.5%        25%
beta[1]   -2.767992112 1.69088184  -6.07500000  -3.893000
beta[2]    0.086004325 0.02936731   0.03000000   0.066150
beta[3]   -0.596305548 0.17785257  -0.96880000  -0.709100
beta[4]    0.056607392 0.01698826   0.02431975   0.045180
beta[5]    1.461678521 1.42021297  -1.35900000   0.515875
beta[6]   -0.009483645 0.03494240  -0.08136150  -0.032150
deviance 106.349330000 3.64375233 101.40000000 103.700000
              50%       75%        97.5%
beta[1]   -2.762000  -1.61500   0.5095000
beta[2]    0.085420   0.10500   0.1457025
beta[3]   -0.590500  -0.47540  -0.2627000
beta[4]    0.056220   0.06775   0.0912800
beta[5]    1.467000   2.39800   4.2180000
beta[6]   -0.008191   0.01421   0.0566900
deviance 105.700000 108.20000 115.2000000
```

13.22.1.3 Estimated Density Plots

We now create Figure 13.6, the estimated trauma density plots of the book. The code samples from the prior each time it is run, so the prior densities change slightly with

each run of the code. The posterior samples were saved and are read, so they do not change.

Code 13.22.4.

```
rm(list = ls())
#Read posterior samples of beta
# Assumes they are in a p x t matrix
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("trauma-samp-thin.Rda")
#load("trauma_samp.Rda")
beta
# establish t (number of iterates)
iterates=length(beta[1,])

# define prior hyperparameters
a=c(1.1,3,5.9,1.3,1.1,1.5)
b=c(8.5,11,1.7,12,4.9,5.5)

#Enter Xtilde
Xtp=c(
 1, 25, 7.84, 60, 0,  0,
 1, 25, 3.34, 10, 0,  0,
 1, 41, 3.34, 60, 1, 60,
 1, 41, 7.84, 10, 1, 10,
 1, 33, 5.74, 35, 0,  0,
 1, 33, 5.74, 35, 1, 35)
Xtilde=t(matrix(Xtp,6,6))
Xtilde

# generate posterior samples of ptildes
# Compute posterior linear predictors and probabilities
ptilde_posterior= Xtilde %*% beta
ptilde_posterior=exp(ptilde_posterior)/(1+exp(ptilde_posterior))

# Matrix for plots
par(mfrow=c(2,3))

# plot
# generate prior sample within 2nd call of density
 for(i in 1:6){
post_p=density(ptilde_posterior[i,])
prior_p=density(rbeta(iterates,a[i],b[i]) )
plot(prior_p,main=c(i),xlab="ptilde", ylim=c(0,10), xlim=c(0,1) )
lines(post_p,lty=5)
legend("topright",c("prior","post"),lty=c(1,5))
```

}

In general, the posterior densities can be obtained by sampling from the discrete approximate posterior and smoothing those samples. That would be, perhaps not absolutely necessary, but certainly convenient if the discrete approximation was from, say, importance sampling where the probabilities in the discrete approximation are not all the same.

13.22.2 Predictive Probabilities

Using the posterior samples β^r, $r = 1,\ldots,t$, to find predictive probabilities at locations x_{fi}, $i = 1,\ldots,n_f$ we compute $F(x'_{fi}\beta^r)$, $r = 1,\ldots,t$, $i = 1,\ldots,n_f$ and average over r. Here F is the cumulative distribution function that defines our binomial sampling model.

To be more succinct, create

$$X_f \equiv \begin{bmatrix} x'_{f1} \\ \vdots \\ x'_{fn_f} \end{bmatrix} \equiv [X_{f1} \quad \cdots \quad X_{fp}]$$

and

$$B \equiv [\beta^1 \quad \cdots \quad \beta^t] \equiv \begin{bmatrix} b'_1 \\ \vdots \\ b'_p \end{bmatrix}.$$

We need to compute

$$F(X_f B),$$

where the one-dimensional cdf F is being applied to each number in the matrix $X_f B$. We then compute the row means of this $n_f \times t$ matrix.

Computationally, we have a couple ways we can get $X_f B$. Obviously,

$$X_f B = [X_f\beta^1 \quad \cdots \quad X_f\beta^t]$$

which is basically what we did with the O-ring data. For the trauma data we do the computation as

$$X_f B = \sum_{j=1}^{p} X_{fj} b'_j.$$

In R notation, we have B identified as `beta` so b'_j is `beta[j,]`. The computation $X_{fj}b'_j$ is the *outer product* of the vectors X_{fj} and b_j, so we would program it something like `outer(Xfj,beta[j,])`, except we use alternative notations for the columns of X_f, notations based on the names of the variables involved.

Book Figure 13.9 involves 8 curves which would require 8 X_f matrices but many of the entries are redundant, which is why we chose this alternative form for com-

puting XB. Again, the command `outer(x,y)` computes xy'. As always J (or `J`) is a column of 1s.

Code 13.22.5.

```
rm(list = ls())

#Read posterior samples of beta
# Assumes they are in a p x t matrix
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("trauma_samp.Rda")
# establish t (number of iterates)
iterates=length(beta[1,])

# Matrix for plots
par(mfrow=c(2,2))

# determine where predictions are made
ISS=seq(0,75,1)
# create a column of 1s
J=1+ISS-ISS
AGE= 60
RTS=3.34


# TI=0, blunt linear predictor
# the command outer computes xy'
w<- outer(J,beta[1,]) + outer(ISS,beta[2,]) +
outer(RTS*J, beta[3,]) + outer(AGE*J, beta[4,])
#TI=1, linear predictor
ww=  w+  outer(J, beta[5,])  + outer(AGE*J,beta[6,])

# turn linear predictors into probabilities
w=exp(w)/(1+exp(w))
ww=exp(ww)/(1+exp(ww))

# compute posterior means of predictive probabilities
y=rowMeans(w)
yy=rowMeans(ww)

# Plot for Age=60 and RTS=3.34
plot(ISS,y,type="l",xlim=c(0,80),ylab="Pr(death)",
    xlab="ISS",main="a. AGE=60 and RTS=3.34",lty=1)
lines(ISS,yy,type="l",lty=2)
legend("bottomright",c("Blunt","Penetrating"),lty=c(1,2))
```

```
AGE=10
RTS=3.34

# TI=0, blunt linear predictor
w<- outer(J,beta[1,]) + outer(ISS,beta[2,]) +
outer(RTS*J, beta[3,]) + outer(AGE*J, beta[4,])
#TI=1, linear predictor
ww=  w+  outer(J, beta[5,])  + outer(AGE*J,beta[6,])

w=exp(w)/(1+exp(w))
ww=exp(ww)/(1+exp(ww))
# compute posterior means of predictive probabilities
y=rowMeans(w)
yy=rowMeans(ww)
plot(ISS,y,type="l",xlim=c(0,80),ylab="Pr(death)",
    xlab="ISS",main="b. AGE=10 and RTS=3.34",lty=1)
lines(ISS,yy,type="l",lty=2)
legend("bottomright",c("Blunt","Penetrating"),lty=c(1,2))


AGE= 60
RTS=5.47

# TI=0, blunt linear predictor
w<- outer(J,beta[1,]) + outer(ISS,beta[2,]) +
outer(RTS*J, beta[3,]) + outer(AGE*J, beta[4,])
#TI=1, linear predictor
ww=  w+  outer(J, beta[5,])  + outer(AGE*J,beta[6,])

w=exp(w)/(1+exp(w))
ww=exp(ww)/(1+exp(ww))
# compute posterior means of predictive probabilities
y=rowMeans(w)
yy=rowMeans(ww)
plot(ISS,y,type="l",xlim=c(0,80),ylab="Pr(death)",
    xlab="ISS",main="c. AGE=60 and RTS=5.47",lty=1)
lines(ISS,yy,type="l",lty=2)
legend("bottomright",c("Blunt","Penetrating"),lty=c(1,2))


AGE= 10
RTS=5.47

# TI=0, blunt linear predictor
```

```
w<- outer(J,beta[1,]) + outer(ISS,beta[2,]) +
outer(RTS*J, beta[3,]) + outer(AGE*J, beta[4,])
#TI=1, linear predictor
ww=  w+  outer(J, beta[5,])  + outer(AGE*J,beta[6,])

w=exp(w)/(1+exp(w))
ww=exp(ww)/(1+exp(ww))
# compute posterior means of predictive probabilities
y=rowMeans(w)
yy=rowMeans(ww)
plot(ISS,y,type="l",xlim=c(0,80),ylab="Pr(death)",
    xlab="ISS",main="d. AGE=10 and RTS=5.47",lty=1)
lines(ISS,yy,type="l",lty=2)
legend("topleft",c("Blunt","Penetrating"),lty=c(1,2))
```

13.22.3 Inference for Regression Coefficients

This is almost identical to the code for the O-ring data. We just use a different set of saved posterior samples.

Code 13.22.6.

```
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("trauma_samp.Rda")
rowMeans(beta)
apply(beta, 1, mean) # Redundant command
#sdv=c(sd(beta[1,]),sd(beta[2,]))
#sdv
apply(beta, 1, sd) # Redundant command
apply(beta, 1, quantile, probs=c(0.05,0.95))
```

13.3 Diagnostics

13.3.1 Case Deletion Influence Measures

This subsection merely sets notation. `LC` is an R function for evaluating the likelihood contribution $L(\beta|y)$. It gets applied as `LC(beta[,r],x[i,],N[i],y[i])` $= L(\beta^r, y_i)$ where `beta` is the $k \times t$ matrix with columns β^r, `x` is the $n \times k$ model matrix X, so `x[i,]` $= x_i'$, with `N` $= (N_1, \ldots, N_n)$ and `y` $= (y_1, \ldots, y_n)$ being n vectors of the number of trials and successes for case i

Table 13.4 Fitted Trauma Model

	Informative Posterior Summaries Based on informative prior				Maximum Likelihood	
Variable	Estimate	Std. Error	0.05%	0.95%	Estimate	Std. Error
Intercept	−1.79	1.10	−3.54	0.02	−2.73	1.62
ISS	0.07	0.02	0.03	0.10	0.08	0.03
RTS	−0.60	0.14	−0.82	−0.37	−0.55	0.17
AGE	0.05	0.01	0.03	0.07	0.05	0.01
TI	1.10	1.06	−0.66	2.87	1.34	1.33
AGE × TI	−0.02	0.03	−0.06	0.03	−0.01	0.03
	Posterior Summaries Based on diffuse prior					
Variable	Estimate	Std. Error	0.05%	0.95%		
Intercept	−2.81	1.60	−5.34	−0.18		
ISS	0.09	0.03	0.05	0.13		
RTS	−0.59	0.17	−0.86	−0.32		
AGE	0.06	0.02	0.03	0.09		
TI	1.46	1.36	−0.79	3.69		
AGE × TI	−0.01	0.03	−0.07	0.05		

`lc` is a t vector with values $L(\beta^r|y_i)$ =`LC(beta[,r],x[i,],N[i],y[i])`. `qti` is a t vector of $\tilde{q}_{r(i)}$ values. To reduce memory use, the indexing on i is implicit.

13.3.2 Estimative and Predictive Influence: O-rings

This subsection corresponds to both Subsections 13.3.2 and 13.3.3 of the text, but only for the O-ring data.

`D1`, `D2`, and `D` are n vectors of D^{β}_{1i}, D^{β}_{2i}, D^{β}_{i} values. `Dp` and `Df` are n vectors of D^{p}_{i}, D^{f}_{i} values.

EXAMPLE 13.3.1. *O-ring Data.*
Begin by reading in the $k \times t$ matrix with columns β^r.

Code 13.3.1.

```
rm(list = ls())
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("post-samp.Rda")
# enter data
n=23
y=c(1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0)
N=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
tau=c(53,57,58,63,66,67,67,67,68,69,70,70,70,70,72,
73,75,75,76,76,78,79,81)
```

```
iterates=length(beta[1,])

# define likelihood contribution function
LC <- function(beta,x,N,y)
{
LP = sum(beta*x)
lc=((exp(LP)/(1+exp(LP)))**y)*((1-(exp(LP)/(1+exp(LP))))**(N-y))
return(lc)}

# Find s dim. vector Pr(y=1|Y,x_iˆf)
# Xf is predictive model matrix
Xf=matrix(c(rep(1,11),seq(31,51,2)),11)
JJf=exp(Xf %*% beta)
JJf=JJf/(1+JJf)
PPf=rowMeans(JJf)

# Find n vector Pr(y=1|Y,x_i)
# X is model matrix
X=matrix(c(rep(1,n),tau),nrow=n)
JJ=exp(X %*% beta)
JJ=JJ/(1+JJ)
PP=rowMeans(JJ)

# Define dummy vectors for diagnostics
# to be redefined below
D1=rep(1,n)
D2=rep(1,n)
D=rep(1,n)
Dp=rep(1,n)
Df=rep(1,n)
lc=rep(1,iterates)
qti=rep(1,iterates)

# Diagnostics
# outer "for" loop finds diagnostics for each i
for(i in 1:n)
{
# inner "for" loop creates "iterates" length vectors of
# L(\betaˆr|y_i) and unstandardized \qtilde_{r(i)} values,
# with i fixed.  Note \qtilde_{r} =  1/t
for(r in 1:iterates)
{
lc[r]=LC(beta[,r],X[i,],N[i],y[i])
qti[r]=1/lc[r]
}
```

```
# standardize \qtilde_{r(i)}
qti=qti/sum(qti)
D1[i]=log(sum(lc*qti))-sum(log(lc)*qti)
D2[i]=mean(log(lc))-log(sum(lc*qti))
D[i]=D1[i]+D2[i]
PPi=JJ%*% qti
PPfi=JJf%*% qti
Dp[i]=sum( (PP-PPi)*(log(PP/(1-PP)) - log(PPi/(1-PPi))) )
Df[i]=sum( (PPf-PPfi)*(log(PPf/(1-PPf)) - log(PPfi/(1-PPfi))) )
}

# Figure 13.11 in text
par(mfrow=c(2,1),mar=c(5, 5, 4, 2) + 0.1)
plot(seq(1:23),Dp,xlab="index",
      ylab=expression(italic(D[i]^p)),cex.lab=1.5)
plot(seq(1:23),Df,xlab="index",
      ylab=expression(italic(D[i]^f)),cex.lab=1.5)
#mtext(expression(D[1]^beta),side=2,cex=1.5,at=1.1)
par(mfrow=c(1,1),mar=c(5, 4, 4, 2) + 0.1)

# Figure not included in text
par(mfrow=c(2,2),mar=c(5, 5, 4, 2) + 0.1)
plot(seq(1:23),D1,ylab=expression(D["1i"]^beta),
                                    xlab="index")
plot(seq(1:23),D2,ylab=expression(D["2i"]^beta),
                                    xlab="index")
plot(seq(1:23),D,ylab=expression(D[i]^beta),
                                    xlab="index")
par(mfrow=c(1,1))
```

13.3.3 Estimative and Predictive Influence: Trauma

This subsection corresponds to both Subsections 13.3.2 and 13.3.3 of the text, but only for the Trauma data. Coding notation is similar to the previous subsection.

EXAMPLE 13.3.2. *Trauma Data.*
Figure 13.12 contains an index plot of the difference in the predictive probabilities of death, $p(y = 1|Y, x_j) - p(y = 1|Y_{(52)}, x_j)$. We also need to compute the D_i^ps for this example. The numerical values of the D_i^ps from this code differ from the 10,000 importance sample values in the previous edition. (The two larger are similar to the 2,000 importance sample numbers reported in Subsection 13.3.5). I have great faith in the importance sample computations reported in the second edition. Ed is a far better programmer than I am. But I haven't been able to find an error in my

computations, so it remains an open question whether the difference in posterior approximations could cause the difference in diagnostic values. The main thing is that *these methods all identified the same observations as being potentially influential.* Similarly, the actual numbers in Figure 13.12 differ from the previous edition, but the relative sizes look the same.

We begin by changing all of `n, a, b, Xtilde, X` from their values in the corresponding O-ring data program.

Code 13.3.2.

```
rm(list = ls())

# Enter data
Trauma <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TRAUMAa.DAT"),
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TRAUMAa.DAT",
  sep="",col.names=c("ID","Death","ISS","TI","RTS","AGE"))
attach(Trauma)
n=length(ISS)
y=1-Death
N=1+ISS-ISS

setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("trauma_samp.Rda")

iterates=length(beta[1,])



# define likelihood contribution function
LC <- function(beta,x,N,y)
{
LP = sum(beta*x)
lc=((exp(LP)/(1+exp(LP)))**y)*((1-(exp(LP)/(1+exp(LP))))**(N-y))
return(lc)}


# Find n vector Pr(y=1|Y,x_i)
# X is model matrix
X=matrix(c(rep(1,n),ISS,RTS,AGE,TI,AGE*TI),nrow=n)
JJ=exp(X %*% beta)
JJ=JJ/(1+JJ)
PP=rowMeans(JJ)

# Define dummy vectors for diagnostics
# to be redefined below
```

```
D1=rep(1,n)
D2=rep(1,n)
D=rep(1,n)
Dp=rep(1,n)
Pdiff=rep(1,n)
lc=rep(1,iterates)
qti=rep(1,iterates)

# Diagnostics
# outer "for" loop finds diagnostics for each i
for(i in 1:n)
{
# inner "for" loop creates "iterates" length vectors of
# L(\beta^r|y_i) and unstandardized \qtilde_{r(i)} values, i fixed
# Note \qtilde_{r} =  1/t
for(r in 1:iterates)
{
lc[r]=LC(beta[,r],X[i,],N[i],y[i])
qti[r]=1/lc[r]
}
# standardize \qtilde_{r(i)}
qti=qti/sum(qti)
#D1[i]=log(sum(lc*qti))-sum(log(lc)*qti)
#D2[i]=mean(log(lc))-log(sum(lc*qti))
#D[i]=D1[i]+D2[i]
PPi=JJ%*% qti
Dp[i]=sum( (PP-PPi)*(log(PP/(1-PP)) - log(PPi/(1-PPi))) )
#Pdiff[i]=sum( PP-PPi)
}
matrix(c(seq(1:n),Dp),ncol=2)
# cases with the highest Dp values
seq(1:n)[Dp>.3]

# This plot does not appear in the book
plot(seq(1:n),Dp,xlab="index",ylab=expression(italic(D[i]^p)),cex.lab=.85)


# Figure 13.12
i=52
for(r in 1:iterates)
{
lc[r]=LC(beta[,r],X[i,],N[i],y[i])
qti[r]=1/lc[r]
}
# standardize \qtilde_{r(i)}
```

```
qti=qti/sum(qti)
PPi=JJ%*% qti
par(mfrow=c(1,1),mar=c(5, 5, 4, 2) + 0.1)
plot(seq(1:n),PP-PPi,xlab="index",ylab=
expression(paste(Pr,"(","y=1","|",
Y,",",x[j],")",-Pr,"(","y=1","|",
Y[(52)],",",x[j],")" )))
```

13.3.4 Model Checking

Computationally, this is more difficult than the next subsection, so you would be well advised to examine that first. The next section involves computing the density $P(Y) \equiv P(Y|M)$ for known Y and three different models M. This section involves not only computing $P(Y|M)$ for known Y (and just one model) but also sampling from the distribution of $P(Y)$ when Y is random.

We sample from the prior as discussed in the book. *BIDA* discusses a more efficient but more convoluted method for these computations that uses samples from the posterior distribution rather than samples from the prior. When integrating the likelihood over the prior, the prior may put most of its probability on places where the likelihood is close to zero. The numerical value of the integral is largely determined by the range of values where the likelihood is not close to zero. To get an accurate evaluation of the integral one needs to sample the nonzero areas of the likelihood extensively. It may take an extremely large sample size from the prior to get a sufficient number of parameter values in the nonzero likelihood areas. Sampling from the posterior should not have that problem but requires some mathematical gymnastics to get an appropriate evaluation for turning a sample from the posterior into an evaluation of a prior integral.

We sample β^r, $r = 1,\ldots,t$ from the prior to evaluate the marginal density. Back in Subsubsection 13.2.1.6 we saved such samples in the file `beta-samples.Rda` as the data frame

```
beta_frame
```

We construct `beta` as our $k \times t$ matrix of prior samples. We also need to sample Y_{fk}. $k = 1,\ldots,M$ from the marginal density. It simplifies matters to have $t = M$?

We begin with the full data model check for the logistic model. The code is written so that a single change near the end will allow model checks for the probit and complementary log-log models. This is so because of similarities between this code and the Link Selection code of the next subsection.

Code 13.3.3.

```
rm(list = ls())
# While only doing the computation for logistic
# models, we include some code for probic and
```

```
# complementary log-log
logit <- function(p)
{
lg = log(p/(1-p))
return(lg)}

logistic <- function(w)
{
lg = exp(w)/(1+exp(w))
return(lg)}

Gum <- function(w)
{
lg = 1-(exp(-exp(w)))
return(lg)}

cll <- function(p)
{
lg = log(-log(1-p))
return(lg)}

### Fletch used BIDA not LOGLIN3 Priors
#   so I have changed them
#a <- c( 1.6, 1 )
#b <- c( 1, 1.6 )
# LOGLIN3 priors
a <- c( 1, .577 )
b <- c( .577, 1 )
iterates=2000    # increase

ptilde1 <- rbeta(iterates, a[1], b[1])
ptilde2 <- rbeta(iterates, a[2], b[2])
# either
# ptildea=t(cbind(ptilde1,ptilde2))
# or
ptilde=c(ptilde1,ptilde2)
ptilde=t(matrix(ptilde,ncol=2))

Xtilde=matrix(c(1,1,55,75),ncol=2)
Xtildeinv=solve(Xtilde)

# enter data
n=23
y=c(1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0)
N=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
```

```
tau=c(53,57,58,63,66,67,67,67,68,69,70,70,70,70,72,
73,75,75,76,76,78,79,81)
X=matrix(c(rep(1,23),tau),ncol=2)
X

pprobit=pnorm(X %*% Xtildeinv %*% qnorm(ptilde))

plogit=logistic(X %*% Xtildeinv %*% logit(ptilde))

pcll=Gum(X %*% Xtildeinv %*% cll(ptilde))


# define likelihood function
den <- function(p,y,N)
{
fp = prod(c(p^y, (1-p)^(N-y)))
# For nonbinary data
#fp = prod(c(p^y, (1-p)^(N-y),choose(N,y)))
return(fp)}
#choose(N, y) or factorial(N)/(factorial(y)*factorial(N-y))

# Marginal density function
# In Link Selection, computed but not a function
marg <- function(y,N)
{

Mprobitd=rep(1,iterates)
Mlogitd=rep(1,iterates)
Mclld=rep(1,iterates)

for(r in 1:iterates){
Mprobitd[r]=den(pprobit[,r],y,N)
Mlogitd[r]=den(plogit[,r],y,N)
Mclld[r]=den(pcll[,r],y,N)}

Mprobit=mean(Mprobitd)
Mlogit=mean(Mlogitd)
Mcll=mean(Mclld)
# change "return" statement for testing
# alternative links
return(Mlogit)
}
# Everything up to this point has been similar to
# the link selection program
```

```
# Full data model check
# For different links, also change "plogit" term
Yf=rep(1,n)
c=0
for(r in 1:iterates){
for(i in 1:n){
Yf[i] = rbinom(1,N[i],plogit[i,r])}
if (marg(Yf,N) <= marg(y,N))  c = c + 1}
Pvalue=c/iterates
Pvalue
```

On my computer, this program involves waiting a little bit.

To incorporate the outlier tests, replace the ”Full data model check” paragraph at the end of the previous program with the following code. On my computer this code took a bit under 30 minutes to run.

Code 13.3.4.

```
# Full data model and outlier check
# For different links, also change "plogit" term
Yf=rep(1,n)
c=0
cc=rep(0,n)
for(r in 1:iterates){
for(i in 1:n){
Yf[i] = rbinom(1,N[i],plogit[i,r])}
for(k in 1:n){
if (marg(Yf[k],N[k]) <= marg(y[k],N[k]))  cc[k] = cc[k] + 1}
if (marg(Yf,N) <= marg(y,N))  c = c + 1}
Pvalue=c/iterates
Pvalue
PV=cc/iterates
PV
```

Because it takes so long to run, my output was

```
> Pvalue
[1] 0.542
> PV=cc/iterates
> PV
 [1] 0.6615 0.6210 0.5995 0.5255 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
[11] 1.0000 1.0000 0.4180 0.4390 1.0000 1.0000 1.0000 0.3475 1.0000 1.0000
[21] 1.0000 1.0000 1.0000
```

13.3.5 Link Selection

This code is similar to the previous subsection in that it again involves sampling from the prior. *BIDA* discusses a more efficient, but more convoluted, method for these computations that uses samples from the posterior distribution rather than samples from the prior. When integrating the likelihood over the prior, the prior may put most of its probability on places where the likelihood is close to zero. The numerical value of the integral is largely determined by the range of values where the likelihood is not close to zero. To get an accurate evaluation of the integral one needs to sample the nonzero areas of the likelihood extensively. It may take an extremely large sample size from the prior to get a sufficient number of parameter values in the nonzero likelihood areas. Sampling from the posterior should not have that problem but requires some mathematical gymnastics to get an appropriate evaluation for turning a sample from the posterior into an evaluation of a prior integral. An easy way to check whether sampling from the prior is working is to gradually increase the number of iterates and see if the Bayes Factor computation remains stable.

In Subsection 13.3.4 for a single model M we discussed finding

$$P(Y) \equiv P(Y|M),$$

where

$$P(Y|M) = \int L(\beta|Y)\pi(\beta) \doteq \frac{1}{t}\sum_r L(\beta^r|Y),$$

with $\beta^1, \ldots, \beta^t$ a random sample from the *prior*. Here both the likelihood function L and the prior π depend on the model M. In this subsection, we find $P(Y|M_j)$ for three models M_j because in Subsection 13.3.5 of the text we found Bayes factors

$$BF_{jk} = \frac{P(Y|M_j)}{P(Y|M_k)}.$$

Throughout this computation, Y is fixed.

The likelihood is

$$L(\beta|Y) \equiv \prod_{i=1}^{n} \binom{N_i}{y_i} \pi_i^{y_i}(1-\pi_i)^{N_i-y_i}, \qquad \pi_i \equiv F(x_i'\beta).$$

This can be rewritten using vectors where we redefine π from the density $\pi(\cdot)$ into an n vector of probabilities,

$$L(\beta|Y) = \exp\left[Y'\log(\pi) + (N-Y)'\log(1-\pi)\right] \prod_{i=1}^{n} \binom{N_i}{y_i}, \qquad \pi \equiv F(X\beta),$$

where $N \equiv (N_1, \ldots, N_n)'$ and functions on real numbers are applied to matrices elementwise. For 0-1 data, $N = J_n$ (a column of 1s) and $\binom{N_i}{y_i} = 1$ for all i.

The prior on β is determined from the prior on $\tilde{p}$ as $\beta = \tilde{X}^{-1}F^{-1}(\tilde{p})$, so the prior distribution of π in our likelihood computation is determined by

$$\pi = F(X\tilde{X}^{-1}F^{-1}(\tilde{p})).$$

In computing we begin by creating a $k \times t$ matrix `ptilde` with each column a sample from the prior on $\tilde{p}$ and then create a version of π as an $n \times t$ matrix with each column a sample from the prior on π. Depending on the choice of F these are called `plogit`, `pprobit`, or `pcll`. We then evaluate the likelihood for each column of these matrices and average to get $P(Y|M_k)$.

EXAMPLE 13.3.1 CONTINUED. *O-ring Data.*
BIDA gives different (more efficient) code that samples from the posterior rather than from the prior.

Code 13.3.5.

```
rm(list = ls())
logit <- function(p)
{
lg = log(p/(1-p))
return(lg)}

logistic <- function(w)
{
lg = exp(w)/(1+exp(w))
return(lg)}

Gum <- function(w)
{
lg = 1-(exp(-exp(w)))
return(lg)}

cll <- function(p)
{
lg = log(-log(1-p))
return(lg)}

### Fletch used BIDA not LOGLIN3 Priors
#   so I have changed them
#a <- c( 1.6, 1 )
#b <- c( 1, 1.6 )
# LOGLIN3 priors
a <- c( 1, .577 )
b <- c( .577, 1 )
iterates=12000    # increase
```

```
ptilde1 <- rbeta(iterates, a[1], b[1])
ptilde2 <- rbeta(iterates, a[2], b[2])
# either
# ptildea=t(cbind(ptilde1,ptilde2))
# or
ptilde=c(ptilde1,ptilde2)
ptilde=t(matrix(ptilde,ncol=2))

Xtilde=matrix(c(1,1,55,75),ncol=2)
Xtildeinv=solve(Xtilde)

# enter data
n=23
y=c(1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0)
N=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
tau=c(53,57,58,63,66,67,67,67,68,69,70,70,70,70,72,
73,75,75,76,76,78,79,81)
X=matrix(c(rep(1,23),tau),ncol=2)
X

pprobit=pnorm(X %*% Xtildeinv %*% qnorm(ptilde))

plogit=logistic(X %*% Xtildeinv %*% logit(ptilde))

pcll=Gum(X %*% Xtildeinv %*% cll(ptilde))


# define likelihood function
den <- function(p,y,N)
{
fp = prod(c(p^y, (1-p)^(N-y)))
# For nonbinary data
#fp = prod(c(p^y, (1-p)^(N-y),choose(N,y)))
return(fp)}
#choose(N, y) or factorial(N)/(factorial(y)*factorial(N-y))


Mprobitd=rep(1,iterates)
Mlogitd=rep(1,iterates)
Mclld=rep(1,iterates)

for(r in 1:iterates){
Mprobitd[r]=den(pprobit[,r],y,N)
Mlogitd[r]=den(plogit[,r],y,N)
Mclld[r]=den(pcll[,r],y,N)}
```

```
Mprobit=mean(Mprobitd)
Mlogit=mean(Mlogitd)
Mcll=mean(Mclld)

Mprobit/Mlogit    #1.086 importance sampling
Mcll/Mlogit       #1.403 i.s.
Mlogit            #4.398109 -6
```

This part of code runs fast enough even with `iterates=552000` but such a large number might be inappropriate with the case deletions about to be addressed.

Figure 13.13 (book Figure 13.13) contains a simultaneous plot of $BF_{21(i)}$ versus i and $BF_{31(i)}$ versus i with $i = 1, \cdots, 23$ for the O-ring data. The full data Bayes factors are given by the intercept at case index 0. To do case deletions, *add this to the end of previous program*

Code 13.3.6.

```
Mprobit=rep(1,n+1)
Mlogit=rep(1,n+1)
Mcll=rep(1,n+1)

Mprobit[1]=mean(Mprobitd)
Mlogit[1]=mean(Mlogitd)
Mcll[1]=mean(Mclld)

for(i in 1:n){
yy=y[-i]
NN=N[-i]

pprobit=pnorm(X[-i,] %*% Xtildeinv %*% qnorm(ptilde))
plogit=logistic(X[-i,] %*% Xtildeinv %*% logit(ptilde))
pcll=Gum(X[-i,] %*% Xtildeinv %*% cll(ptilde))

for(r in 1:iterates){
Mprobitd[r]=den(pprobit[,r],yy,NN)
Mlogitd[r]=den(plogit[,r],yy,NN)
Mclld[r]=den(pcll[,r],yy,NN)
}

Mprobit[i+1]=mean(Mprobitd)
Mlogit[i+1]=mean(Mlogitd)
Mcll[i+1]=mean(Mclld)
}

# Figure 13.13 in text
```

```
plot(seq(0,n),Mcll/Mlogit,xlab="Index",ylab="Bayes Factor",
ylim=c(1,1.5),type="l",lty=2)
lines(seq(0,n),Mprobit/Mlogit)
legend("left",c("Log-Log","Probit"),lty=c(2,1))
```

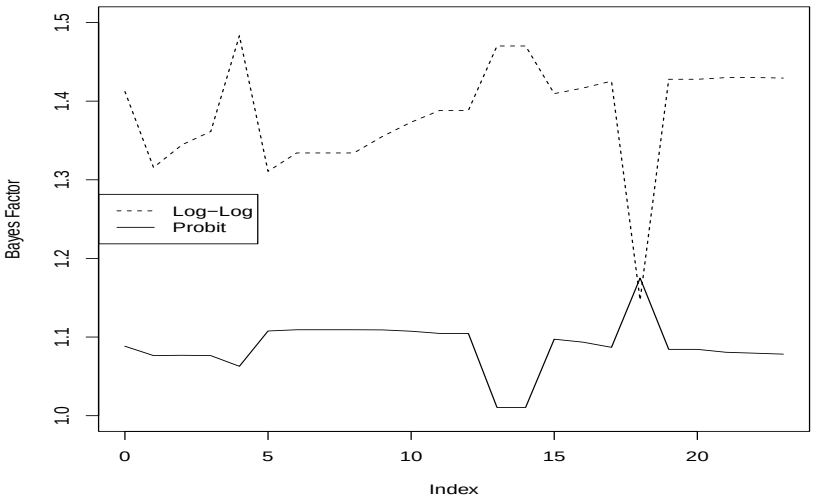


Fig. 13.12 O-Ring Data: Bayes Factors with Case Deletion

EXAMPLE 13.3.2 CONTINUED. *Trauma Data*.
From textbook: $BF_{21} = 1.05$, $BF_{13} = 20.72$. *It is left as an exercise to modify the code as necessary.* In particular change the vector $\tilde{p}$ and matrices X and $\tilde{X}$.

13.3.6 Sensitivity Analysis

The following code merely illustrates a theoretical fact. Since our prior is a DAP (data augmentation prior), we begin by repeating the code for fitting with flat priors but augment our data with the prior observations. This begins with changing the BUGS model file `trauma_flat.txt` into `trauma_flat_binomial.txt` to allow for binomial sampling rather than Bernoulli sampling.

```
model{
 for(i in 1:n){
   Death[i] ~ dbin(p[i],N[i])
   logit(p[i]) <- beta[1] + beta[2]*ISS[i] + beta[3]*RTS[i]
```

```
                    + beta[4]*AGE[i] + beta[5]*TI[i]
                    + beta[6]*AGE[i]*TI[i]
 }
 for(i in 1:6){ beta[i] ~ dflat() }
}
```

In the code for running this BUGS program in R we need to define the binomial sample size variable `N`, augment the `data` statement with *N*, change the input file in `bugs`, and augment the data.

Code 13.3.7.

```
rm(list = ls())
# enter data size
n=300
iterates=80000
burn_in=5000
library(R2OpenBUGS)
BUGS_path <-
"c:\\Program Files (x86)\\OpenBUGS\\OpenBUGS323\\OpenBUGS.exe"
setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
working_dir <- "c:\\E-drive\\Books\\LOGLIN3\\BAYES\\"

# Enter data
Trauma <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TRAUMAa.DAT"),
"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TRAUMAa.DAT",
  sep="",col.names=c("ID","Death","ISS","TI","RTS","AGE"))
attach(Trauma)
Death=1-Death

# Enter prior information
a=c(1.1,3,5.9,1.3,1.1,1.5)
b=c(8.5,11,1.7,12,4.9,5.5)

#Enter Xtilde
Xtp=c(
 1, 25, 7.84, 60, 0,  0,
 1, 25, 3.34, 10, 0,  0,
 1, 41, 3.34, 60, 1, 60,
 1, 41, 7.84, 10, 1, 10,
 1, 33, 5.74, 35, 0,  0,
 1, 33, 5.74, 35, 1, 35)
Xtilde=t(matrix(Xtp,6,6))
Xtilde

# Turn prior into augmented data
```

```
N=rep(1,n)
n=n+6
N=c(N,b+a)
Death=c(Death,a)
ISS=c(ISS,Xtilde[,2])
RTS=c(RTS,Xtilde[,3])
AGE=c(AGE,Xtilde[,4])
TI=c(TI,Xtilde[,5])

# new "data" includes "N"
data <- list( "n","N", "Death","ISS","RTS","AGE","TI")

inits <- function() {
  list(beta=c(0.5,0.5,0.5,0.5,0.5,0.5))
}

parameters <- list( "beta" )

# Change "model.file"
Trauma.sim <- bugs( data, inits, parameters,
                   model.file="trauma_flat_binomial.txt",
                    n.chains=1, n.iter=iterates+burn_in,
                    n.thin=1, n.burnin=burn_in,
                    OpenBUGS.pgm=BUGS_path,
                    working.directory=working_dir,
                    debug=F )
Trauma.sim$summary
# save p x t matrix of gamma posterior iterates
#beta <- t(Trauma.sim$sims.list$beta)
#save(beta,file="trauma_flat_samp.Rda")
```

Theoretically, this program should give the same results as our informative prior and the actual results to be within sampling variation of the results given earlier.

```
                mean         sd         2.5%         25%
beta[1]   -1.76639352 1.13577270  -4.01202500  -2.531000
beta[2]    0.06516500 0.02156228   0.02350000   0.050620
beta[3]   -0.60002985 0.14309518  -0.88840000  -0.694025
beta[4]    0.04743730 0.01380904   0.02070000   0.038100
beta[5]    1.09188298 1.09893998  -1.08600000   0.351000
beta[6]   -0.01726508 0.02786374  -0.07196025  -0.036010
deviance 121.39621875 3.55173482 116.50000000 118.800000
               50%         75%        97.5%
beta[1]    -1.75900  -0.9964000    0.4328025
beta[2]     0.06484   0.0794700    0.1082000
beta[3]    -0.59770  -0.5024000   -0.3264000
```

```
beta[4]     0.04726   0.0566300    0.0753700
beta[5]     1.09900   1.8400000    3.2240000
beta[6]    -0.01695   0.0018155    0.0363400
deviance  120.70000 123.3000000  130.0000000
```

Now that we know this works, we can delete the augmented data, one at a time, to check the sensitivity of the analysis to each prior observation.

Since the DAP is equivalent to adding data with a flat prior, we can check the sensitivity of the analysis by deleting, in turn, the augmenting observations, just like we did the real observations in Figure 13.12 of the book. The following code is just a modification of code for Figure 13.12 that in turn drops out each of the six prior observations. (The code also eliminates some diagnostic computations from the earlier program that are not needed here.)

Code 13.3.8.

```
rm(list = ls())

# Enter data
Trauma <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TRAUMAa.DAT"),
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TRAUMAa.DAT",
  sep="",col.names=c("ID","Death","ISS","TI","RTS","AGE"))
attach(Trauma)
n=length(ISS)
y=1-Death
N=1+ISS-ISS

setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
load("trauma_samp.Rda")

iterates=length(beta[1,])

# Enter prior information
a=c(1.1,3,5.9,1.3,1.1,1.5)
b=c(8.5,11,1.7,12,4.9,5.5)

#Enter Xtilde
Xtp=c(
 1, 25, 7.84, 60, 0,  0,
 1, 25, 3.34, 10, 0,  0,
 1, 41, 3.34, 60, 1, 60,
 1, 41, 7.84, 10, 1, 10,
 1, 33, 5.74, 35, 0,  0,
 1, 33, 5.74, 35, 1, 35)
Xtilde=t(matrix(Xtp,6,6))
Xtilde
```

```
# define likelihood contribution function
LC <- function(beta,x,N,y)
{
LP = sum(beta*x)
lc=((exp(LP)/(1+exp(LP)))**y)*((1-(exp(LP)/(1+exp(LP))))**(N-y))
return(lc)}


# Find n vector Pr(y=1|Y,x_i)
# X is model matrix
X=matrix(c(rep(1,n),ISS,RTS,AGE,TI,AGE*TI),nrow=n)
JJ=exp(X %*% beta)
JJ=JJ/(1+JJ)
PP=rowMeans(JJ)

# Define dummy vectors for diagnostics
# to be redefined below
Pdiff=rep(1,n)
lc=rep(1,iterates)
qti=rep(1,iterates)

# Figure 13.14
par(mfrow=c(3,2))#,mar=c(5, 5, 4, 2) + 0.1)
i=1
for(r in 1:iterates)
{
lc[r]=LC(beta[,r],Xtilde[i,],b[i]+a[i],a[i])
qti[r]=1/lc[r]
}
# standardize \qtilde_{r(i)}
qti=qti/sum(qti)
PPi=JJ%*% qti

plot(seq(1:n),PP-PPi,xlab="index",ylab=
expression(paste(Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y),")",-Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y)[(1)],")" )),main="Omit location 1")

i=2
for(r in 1:iterates)
{
lc[r]=LC(beta[,r],Xtilde[i,],b[i]+a[i],a[i])
qti[r]=1/lc[r]
}
```

```
# standardize \qtilde_{r(i)}
qti=qti/sum(qti)
PPi=JJ%*% qti

plot(seq(1:n),PP-PPi,xlab="index",ylab=
expression(paste(Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y),")",-Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y)[(2)],")" )),main="Omit location 2")


i=3
for(r in 1:iterates)
{
lc[r]=LC(beta[,r],Xtilde[i,],b[i]+a[i],a[i])
qti[r]=1/lc[r]
}
# standardize \qtilde_{r(i)}
qti=qti/sum(qti)
PPi=JJ%*% qti

plot(seq(1:n),PP-PPi,xlab="index",ylab=
expression(paste(Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y),")",-Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y)[(3)],")" )),main="Omit location 3")

i=4
for(r in 1:iterates)
{
lc[r]=LC(beta[,r],Xtilde[i,],b[i]+a[i],a[i])
qti[r]=1/lc[r]
}
# standardize \qtilde_{r(i)}
qti=qti/sum(qti)
PPi=JJ%*% qti

plot(seq(1:n),PP-PPi,xlab="index",ylab=
expression(paste(Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y),")",-Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y)[(4)],")" )),main="Omit location 4")

i=5
for(r in 1:iterates)
{
lc[r]=LC(beta[,r],Xtilde[i,],b[i]+a[i],a[i])
qti[r]=1/lc[r]
```

```
}
# standardize \qtilde_{r(i)}
qti=qti/sum(qti)
PPi=JJ%*% qti

plot(seq(1:n),PP-PPi,xlab="index",ylab=
expression(paste(Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y),")",-Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y)[(5)],")" )),main="Omit location 5")

i=6
for(r in 1:iterates)
{
lc[r]=LC(beta[,r],Xtilde[i,],b[i]+a[i],a[i])
qti[r]=1/lc[r]
}
# standardize \qtilde_{r(i)}
qti=qti/sum(qti)
PPi=JJ%*% qti

plot(seq(1:n),PP-PPi,xlab="index",ylab=
expression(paste(Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y),")",-Pr,"(","y=1","|",
Y,",",x[j],",",tilde(Y)[(6)],")" )),main="Omit location 6")
```

13.4 Posterior Computations

No importance sampling code was written for R.

13.5 A Log-Linear Model with Over Dispersion (Random Effects)

For ease of programming in the log-linear model for Aché hunting, $(\beta_0, \beta_1, \beta_2) \equiv$ `(beta[1],beta[2],beta[3])`. The following BUGS model for the Aché hunting data was saved as `ache-m.txt`.

```
model{
 for(k in 1:38){
   kills[k] ~ dpois(lambda[k]*trials[k])
   log(lambda[k]) <- beta[1]+beta[2]*(age[k]-mean(age[]))+
   beta[3]*(age[k]-mean(age[]))**2+eta[k] }
```

```
 for(i in 1:38){ eta[i] ~ dnorm(0,tau) }
 for(i in 1:11){
    log(ExKill[i]) <-  beta[1] + beta[2]*(pages[i]-mean(age[])) +
    beta[3]*(pages[i]-mean(age[]))**2 + 1/(2*tau)
   d[i] ~ dnorm(0,tau)
   log(r[i]) <- beta[1] + beta[2]*(pages[i]-mean(age[])) +
   beta[3]*(pages[i]-mean(age[]))**2  + d[i]
 }
 beta[1] ~ dnorm(0,0.0001)
 beta[2] ~ dnorm(0,0.0001)
 beta[3] ~ dnorm(0,0.0001)
 tau ~ dgamma(0.0001,0.0001)
}
```

The "prediction" `for` loop that goes from 1 to 11 near the end could have been performed in R from the simulated values for `beta` and `tau` be we decided to incorporate the predictions into the BUGS model.

What follows is my program (modified from *BIDA*) to save the posterior samples, give the standard summaries and produce Figures 13.15 and 13.16 in the book. I decided to use JAGS for this simulation. Most of my data files do not contain column names but this one does, so the `read.table` command includes `header=T` rather than a `col.names` command.

Code 13.5.1.

```
rm(list = ls())
# Enter data
Ache <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB13-5.DAT"),
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TAB13-5.dat",
  sep="",header=T)
attach(Ache)
Ache

library(R2jags)

setwd("c:\\E-drive\\Books\\LOGLIN3\\BAYES\\")
working_dir <- "c:\\E-drive\\Books\\LOGLIN3\\BAYES\\"

# Larger burn_in  and iterates for
# highly correlated data
iterates <- 50000
burn_in <- 10000

inits <- function() {
  list( beta = c( 0, 0, 0),tau=1 )
}
```

```
ache.dat.jags<-list("kills","age","trials","pages")
ache.params<-c("beta","tau","ExKill","r")

ache.fit <- jags(data = ache.dat.jags, inits = inits,
  parameters.to.save = ache.params, n.chains = 1, n.thin=1,
n.iter = iterates+burn_in, n.burnin = burn_in,
model.file = "ache-m.txt")

# Rename Output
beta.a <- t(ache.fit$BUGSoutput$sims.list$beta)
tau.a = ache.fit$BUGSoutput$sims.list$tau
ExKill.a = ache.fit$BUGSoutput$sims.list$ExKill
r.a = ache.fit$BUGSoutput$sims.list$r

# Save Output
save(beta.a,file="ache-beta.Rda")
save(tau.a,file="ache-tau.Rda")
save(ExKill.a,file="ache-ExKill.Rda")
save(r.a,file="ache-lambda.Rda")

#Table of Coefficients for Model Parameters
# in book Table 13.6.
rowMeans(beta.a)
apply(beta.a, 1, mean) # Redundant command
apply(beta.a, 1, sd)
t(apply(beta.a, 1, quantile, probs=c(.5,0.025,0.975)))

sigma.a=1/sqrt(tau.a)
mean(sigma.a)
apply(sigma.a, 2, mean) # Redundant command
sd(sigma.a)
apply(sigma.a, 2, sd) # Redundant command
t(apply(sigma.a, 2, quantile, probs=c(.5,0.025,0.975)))

# Standard jags Summaries
summary(ache.fit$BUGSoutput$sims.list$beta)
summary(t(beta.a))
summary(ache.fit$BUGSoutput$sims.list$tau)
summary(ache.fit$BUGSoutput$sims.ist$ExKill)
summary(ache.fit$BUGSoutput$sims.list$r)

# Code for Figure 13.15
Aplot = t(apply(ExKill.a,2 , quantile, probs=c(.5,0.025,0.975)))
plot(pages,Aplot[,3],type="l",lty=1,lwd=2,xlab="Age",
```

```
ylab="Expected Kills",ylim=c(0,.8))
lines(pages,Aplot[,1],lty=2,lwd=2)
lines(pages,Aplot[,2],lty=1,lwd=2)

# Code for Figure 13.16
AAplot = t(apply(r.a,2 , quantile, probs=c(.5,0.025,0.975)))
plot(pages,AAplot[,3],type="l",lty=1,lwd=2,xlab="Age",
ylab="Latent Variable",ylim=c(0,1.3))
lines(pages,AAplot[,1],lty=2,lwd=2)
lines(pages,AAplot[,2],lty=1,lwd=2)
```

Again, we could have computed Figures 13.16 and 13.17 from the saved values in `beta.a` and `tau.a` but we programmed them into the BUGS model in `ache-m.txt`

13.5.1 Contingency Tables

Specifying priors for contingency tables can get complicated. Actually fitting the models is similar to the methods illustrated.

One simplistic approach is to create a DAP (data augmentation prior) by essentially adding prior observations to every entry in the table. For a multinomial data sample a DAP can be accomplished by using a Dirichlet prior for the cell probabilities with Dirichlet parameters that are the number of prior observations. (Just as the multinomial generalizes the binomial, the Dirichlet generalizes the beta.) For product-multinomial sampling, one could put independent Dirichlet priors on the probabilities of the independent multinomials. Simple Dirichlet priors have been found quite limiting by many people in that the cell prior probabilities have to be "almost" independent. (You get Dirichlet probabilities by generating independent Gamma distributions and dividing by their sum.) More sophisticated priors can be developed by making conditional statements about the probabilities within a multinomial.

Example 15.1.1 and Table 15.1 of the book, as well as Christensen (1996, Section 8.5; 2015, Section 5.5) considers data from Lazerwitz (1961) on occupations and religions. The actual frequentist analysis differs little between whether the data are considered one multinomial sample or three independent samples, one for each religious group, but the interpretation of that analysis depends on the sampling scheme because the meanings of the parameters depend on the sampling scheme. The prior distributions also depend on the meaning of the parameters. For product multinomial sampling, we need a prior distribution for each probability of a Roman Catholic falling into the four occupation categories. Incorporating old prejudices about large numbers of Italian and Irish immigrants being RC and lower class, we might use a Dirich$(0.5, 0.5, 1.5, 2.5)$ associated with 5 prior observations and best prior guesses for the category probabilities of $(0.1, 0.1, 0.3, 0.5)$. Again, using old stereotypes, we might use a prior for Protestants of Dirich$(1, 1, 1.5, 1.5)$ associated

with 5 prior observations and best prior guesses for the category probabilities of $(0.2, 0.2, 0.3, 0.3)$ and for Jewish folks, prior guesses of $(0.32, 0.33, 0.25, 0.1)$ with 2 prior observations for a Dirich$(0.64, 0.66, 0.5, 0.2)$.

BUGS model. **Work out a simple program** `occupation-m.tex`

```
model{
or[] ~ dmult(pr[],Nr)
op[] ~ dmult(pj[],Np)
oj[] ~ dmult(pj[],Nj)

  pr[] ~ ddirich(ar[])
  pp[] ~ ddirich(ap[])
  pj[] ~ ddirich(aj[]) }
```

Code 13.5.2.

```
lazer <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB15-1.DAT"),
# "C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-6.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TAB15-1.dat",
  sep="",col.names=c("O","Rel","Occ"))
attach(lazer)
lazer
laz <- xtabs(O~Rel+Occ)
laz

burn_in=1000
iterates=10000

or=O[Rel==1]
op=O[Rel==2]
oj=O[Rel==3]
Nr=sum(or)
Np=sum(op)
Nj=sum(oj)
ar=c(0.5,0.5,1.5,2.5)
ap=c(1,1,1.5,1.5)
aj=c(0.64,0.66,0.5,0.2)

occ.dat.jags<-list("or","op","oj","ar","ap","aj","Nr","Np","Nj")
occ.params<-c("pr","pp","pj")
inits <- function() {
  list( pr = ar/sum(ar), pp = ap/sum(ap), pj = aj/sum(aj))
}

occ.fit <- jags(data = occ.dat.jags, inits = inits,
  parameters.to.save = occ.params, n.chains = 1, n.thin=1,
```

```
n.iter = iterates+burn_in, n.burnin = burn_in,
model.file = "occupation-m.txt")



 x[] ~ dmulti(p[], N)
  p[] ~ ddirich(alpha[])
  ddirch
for (i in 1:I) {
x[i,1:K] ~ dmnorm(mu[], tau[,])
}

yr[] ~ dmulti(pr[],



model{
 for(k in 1:38){
   kills[k] ~ dpois(lambda[k]*trials[k])
   log(lambda[k]) <- beta[1]+beta[2]*(age[k]-mean(age[]))+
   beta[3]*(age[k]-mean(age[]))**2+eta[k] }
 for(i in 1:38){ eta[i] ~ dnorm(0,tau) }
 for(i in 1:11){
    log(ExKill[i]) <-  beta[1] + beta[2]*(pages[i]-mean(age[])) +
    beta[3]*(pages[i]-mean(age[]))**2 + 1/(2*tau)
   d[i] ~ dnorm(0,tau)
   log(r[i]) <- beta[1] + beta[2]*(pages[i]-mean(age[])) +
   beta[3]*(pages[i]-mean(age[]))**2  + d[i]
 }
 beta[1] ~ dnorm(0,0.0001)
 beta[2] ~ dnorm(0,0.0001)
 beta[3] ~ dnorm(0,0.0001)
 tau ~ dgamma(0.0001,0.0001)
}
```

13.6 OpenBUGS GUI

The reason for running the *GUI (Graphical User Interface)* is that it makes determining Markov chain convergence easier and provides flexibility in determining chain size.

The price you pay for the added flexibility of using the OpenBUGS GUI is that you have to know how to identify the model, the data, the parameters, and the parameters' initial values within the GUI (by clicking appropriate buttons) whereas

all these things are explicit in `R2OpenBUGS`. You still have to write a program to run OpenBUGS through the GUI but you only specify the pieces in the program whereas you identify what the pieces are/mean in the GUI rather than the program. In a program written for the OpenBUGS GUI the statement of the model is exactly as before and we need to add statements to identify the data and specify initial values.

The GUI process will require trips to the menu options `Model`, then `Inference`, back to `Model`, then back to `Inference`. To specify all the parts of the model, you will need to highlight parts of the program before hitting appropriate buttons.

Having copied the code into OpenBUGS, the next step is to open the `Model` menu and select the `Specification` tool. You will need to hit the buttons

1. check model,
2. load data,
3. compile,
4. load inits (for initial values)

but before hitting the `check model` button or a `load` button you need to ***highlight*** *the appropriate `model` or `list` statement in your GUI program!*

Our goal here is to reproduce the Uncentered Data results from from Subsection 13.2.1.4. This uses the model in `Oring_model_a.txt` from Subsection 13.2.1.3. In particular, we need to copy the data and initial values into the GUI (we cannot read the data in). To facilitate that, I constructed the file `Oring_model_a_GUI.txt` as follows:

```
# model from Oring_model_a.txt
model{
  for( i in 1:23){
  y[i]  ~ dbin( p[i], 1)
  logit( p[i] ) <- beta[1] + beta[2] * temp[i]
  }
  for( j in 1:2 ){
  ptilde[j]  ~ dbeta( a[j], b[j] )
  }
  beta[1] <- (75/20) * logit( ptilde[1] )
                - (55/20) * logit( ptilde[2] )
  beta[2] <- (-1/20) * logit( ptilde[1] )
                + (1/20) * logit( ptilde[2] )}

# data redefined
list(y=c(0,1,0,0,0,0,0,0,1,1,1,0,0,1,0,0,0,0,0,
0,0,1,0,1),
temp=c(66,70,69,68,67,72,73,70,57,63,70,78,67,
53,67,75,70,81,76,79,75,76,58),
a=c(1.000,0.577), b=c(0.577,1.000))
```

```
# initial values (inits) redefined
list( ptilde = c( 0.5, 0.5 ) )
```

Note that in the statement `model for (1 in 1:23)` I have replaced `n` with 23 because I do not have the luxury of having R tell me how many observations I have. To create this file I reran the earlier R code, printed out relevant items, edited them, and saved the results.

Having installed OpenBUGS, click or double click the OpenBugs icon.

1. Go to the `File` menu and click on `New`. This opens a window entitled `untitled1`. Within the `File` menu, hit `Save As` and save the file as `Oring_GUI.odc` in an appropriate folder, see Figure 13.13.

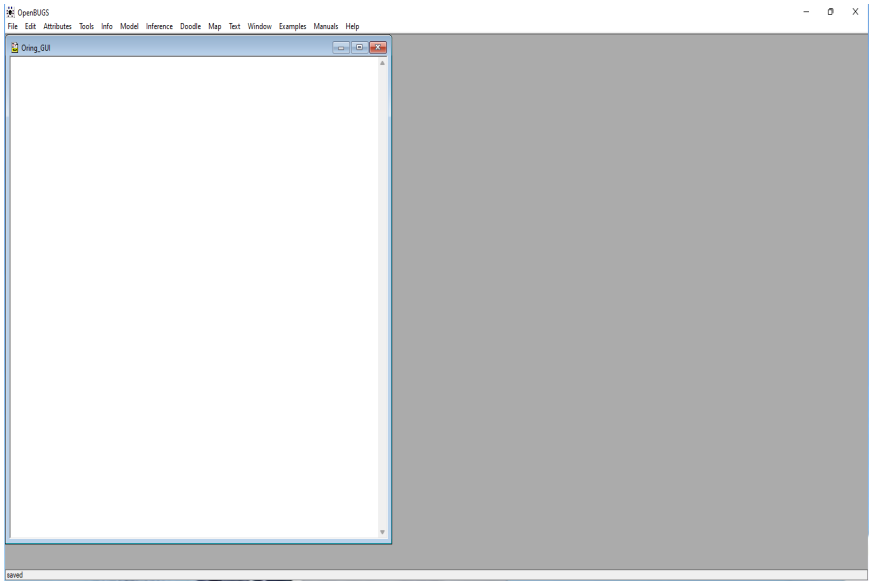


Fig. 13.13 Screenshot of OpenBUGS GUI.

2. In the `Oring_GUI` window, copy the contents of `Oring_model_a_GUI.txt`, see Figure 13.14. (I did some other things like make the window larger and increase the type size.)

3. Open the `Model` menu and click on `Specification...` This opens a new "Specification Tool" window, see Figure 13.15.

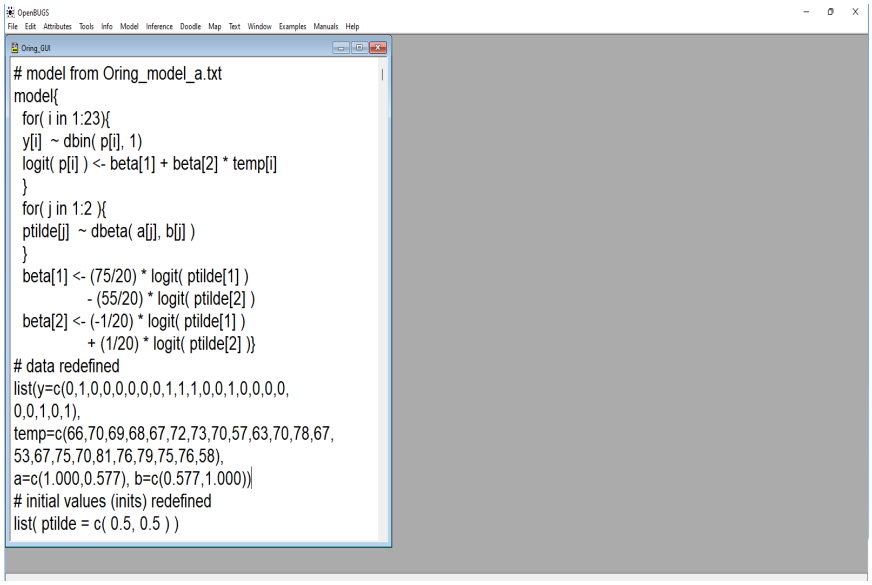


Fig. 13.14 OpenBUGS screenshot of GUI with modeling information.

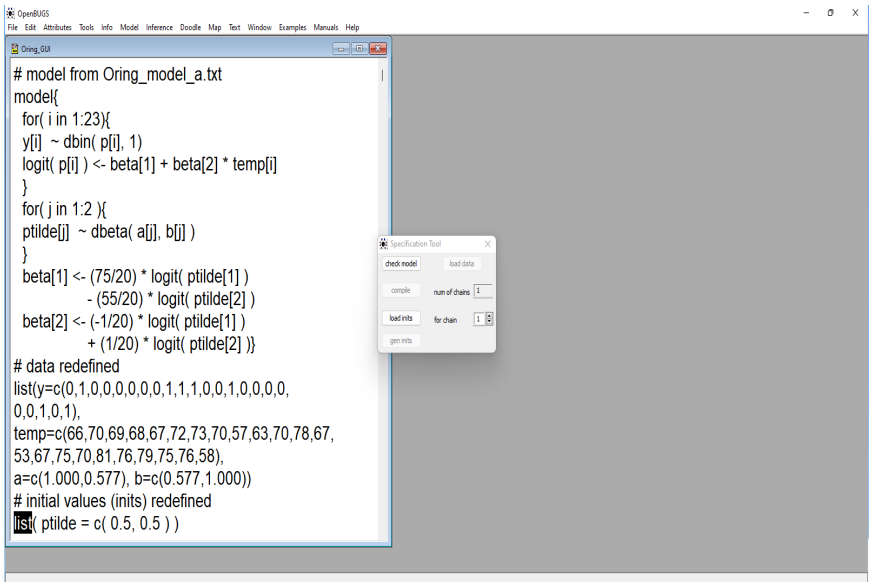


Fig. 13.15 OpenBUGS screenshot of GUI with Specification Tool window.

We will need to go back and forth between the "Specification Tool" and `Oring_GUI` windows. In the bottom left of the OpenBUGS window, it will tell you what is going on, that is, whether things are working properly and, if not, what OpenBUGS thinks is going wrong. OpenBUGS uses a vertical line as a cursor, and if something is wrong, OpenBUGS puts a box around where it thinks the problem exists. (We frequently manage to fool OpenBUGS with our errors.)

4. Double click in the middle of the word "`model`" in `Oring_GUI` then click on check model in the "Specification Tool" window.
5. Double click in the middle of the word "`list`" in the line that contains the data in `Oring_GUI` then click on load data in "Specification Tool."
6. Click on compile in "Specification Tool."
7. Double click in the middle of the word "`list`" in the line with the initial values in `Oring_GUI` then click on load inits in "Specification Tool."
8. Normally you can kill the "Specification Tool" window at this point.
9. Back in the OpenBUGS window, open the `Inference` menu and choose `Samples...` which opens the "Sample Monitor Tool" window, see Figure 13.16.

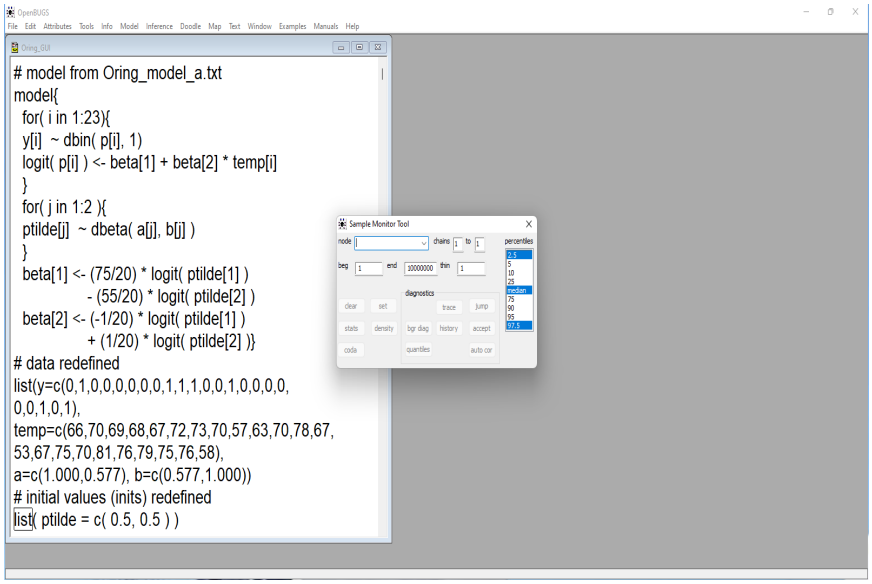


Fig. 13.16 OpenBUGS screenshot of GUI with "Sample Monitor Tool" window.

10. In the box by `node` enter `beta` and click on the set button. This box tells OpenBUGS what quantities (parameters) you want to evaluate in your analysis. To tell OpenBUGS you are done entering nodes, put an asterisk in the box.
11. Go back to the `Model` menu and choose `Update...` to open the "Update Tool" window.
12. Go to "Update Tool," cf., Figure 13.17. In the box next to `updates` change the number to 11000 which is the number of iterates plus the burn-in that we want. Click the update button in "Update Tool." When the number in the box next to `iteration` reaches 11000, go back to the "Sample Monitor Tool."

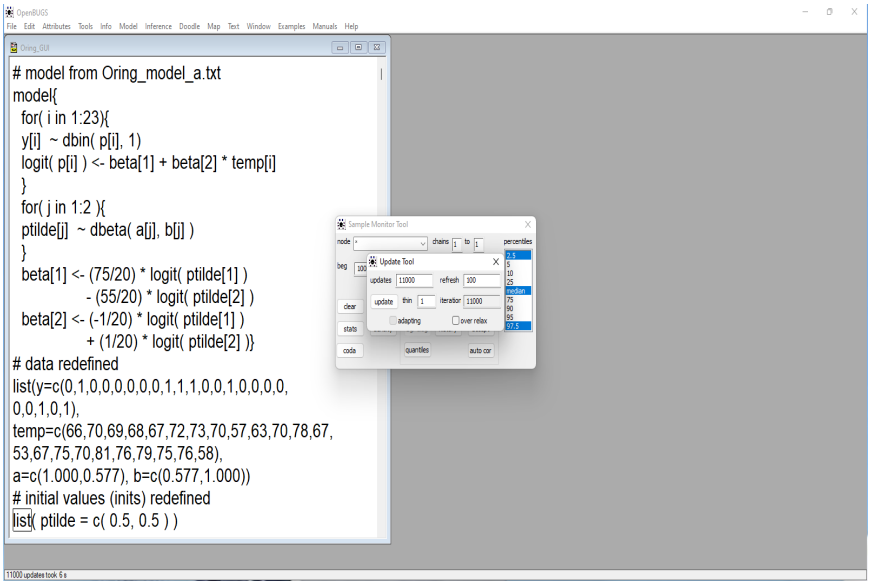


Fig. 13.17 OpenBUGS screenshot of GUI with "Update Tool" window.

13. A "burn in" value can be specified in the `beg` box of the "Sample Monitor Tool." Type "1001" in this box. We are throwing out the first 1000 iterates to eliminate any effect of our starting values on posterior inferences.
14. In the "Sample Monitor Tool" click on stats and density. I moved the windows before taking the screen shot in Figure 13.18.

15. In the "Sample Monitor Tool" click on history and auto cor. I deleted the "Node statistics" and "Posterior density" windows and moved the windows before taking the screen shot in Figure 13.19.

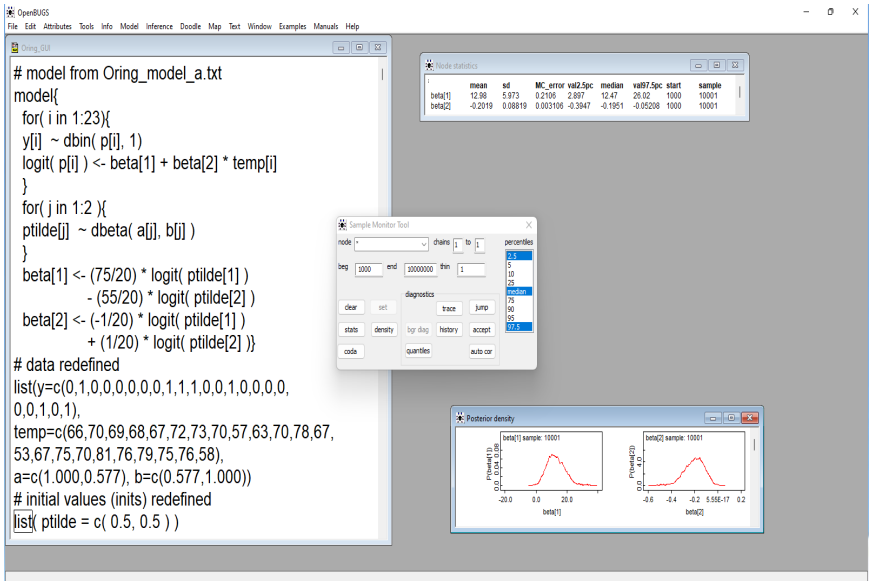


Fig. 13.18 OpenBUGS screenshot of GUI with table of coefficients and densities.

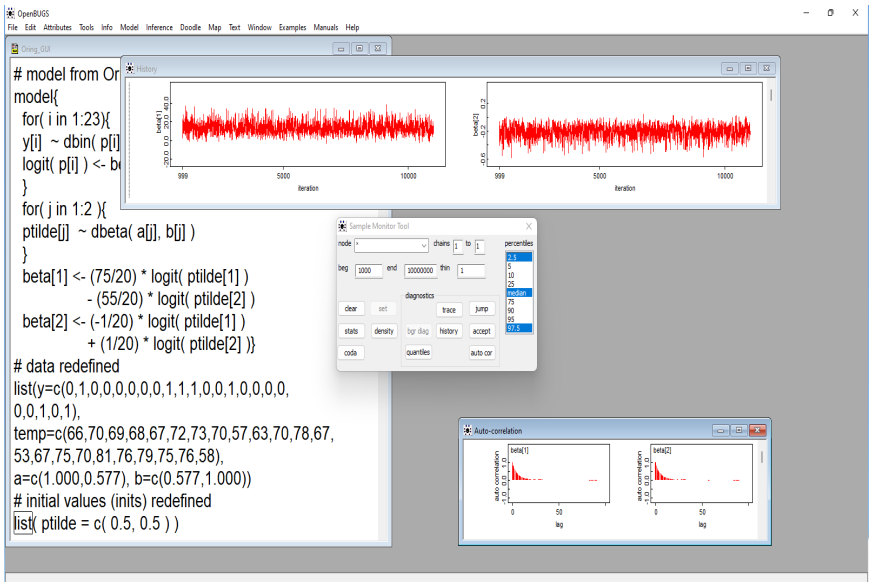


Fig. 13.19 OpenBUGS screenshot of GUI with diagnostics.

16. If you want to save your results, go to the File menu and Save them.

In our opinion (you can guess who ”we” are), the two most important things the GUI adds is the ability to click on the `history` and the `auto cor` buttons in the `Sample Monitor Tool`. For the McMC process to be working, we need the Markov chain to converge to a stationary process in the standard sense of stationarity used for Time Series Analysis, e.g. Christensen (2019, Section 6.1 and Chapter 7). The command `history` plots the McMC time series so you can look to see if it seems stationary after a suitable burn-in period. “Stationary” intuitively means that the the process stays the same. No matter where in the sequence you start to look at it, the past should always look the same and the future should always look the same. Of course, this is a property of the theoretical random process. We only get one, or perhaps a few, looks at the process from which to decide if it looks close enough to being stationary. For these applications, with large sample sizes from the chain, one way to think of it is that the history should pretty well fill up a rectangular box. The autocorrelation function (ACF) needs to pass what I think of as the “ink test.” If you see a lot of red ink (or pixels or toner) in OpenBUGS' `auto cor` plot, you likely have a problem with stationarity. Problems with high correlations (like the uncentered O-ring data), typically require larger sample sizes, which extends the horizontal axis of the ACF which typically reduces the amount of ink needed to graph it in the same horizontal length. So if the process is stationary, even with a lot of correlation, if you run the process long enough, the ACF should pass the ink test. If the ink reduces linearly, rather than exponentially (even with large correlations), you likely have a problem with stationarity.

While thinning the Markov chain is typically not necessary for most inferential procedures, it seems to me that thinning can be a valuable tool in checking whether the Markov chain has converged to a stationary process. If the entire (converged) process is stationary, any thinned process has to be stationary. If you can find a thinned process that does not look stationary, the entire process is unlikely to be stationary. With large sample sizes, the history plot can get so compressed as to lose the detail necessary for detecting nonstationarity. With sufficient thinning, a stationary chain should look similar to white noise (i.e., independent and identically distributed).

Chapter 14
Exact Conditional Tests

The section General Theory has some discussion of general computing.

14.1 Two-Factor Tables

Exact conditional tests for two-way tables can be performed in R using `fisher.test`. As shown in the next section, I used `fisher.test` to check my algebra on the 2×4 example table (and found some mistakes to correct). Some inconsistencies of method for 2×2 tables that exist in `fisher.text` are addressed by the new package `exact2x2`.

I computed the probabilities for the 3×3 example by hand but code for computing the test statistics follows.

Code 14.1.1.

```
rm(list = ls())
# indices for fitting glm
row=c(1,1,1,2,2,2,3,3,3)
col=c(1,2,3,1,2,3,1,2,3)
row=factor(row)
col=factor(col)
# These are the 12 tables
# I read them in by row
t1=c(1,0,0,0,0,2,0,2,1)
t2=c(1,0,0,0,1,1,0,1,2)
t3=c(1,0,0,0,2,0,0,0,3)
t4=c(0,0,1,0,0,2,1,2,0)
t5=c(0,0,1,0,1,1,1,1,1)
t6=c(0,0,1,0,2,0,1,0,2)
t7=c(0,1,0,0,0,2,1,1,1)
t8=c(0,1,0,0,1,1,1,0,2)
```

```
t9=c(0,0,1,1,0,1,0,2,1)
t10=c(0,0,1,1,1,0,0,1,2)
t11=c(0,1,0,1,1,0,0,0,3)
t12=c(0,1,0,1,0,1,0,1,2)
# "matrix" reads by column
# so they need to be transposed

# for each group of three commands
# first fits the model in glm to compute G^2
# second fits the model in chisq.test to compute X^2
# then prints out the numbers X^2, G^2
fit1 = glm(t1 ~ row + col, poisson)
f1=chisq.test(t(matrix(t1,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t2 ~ row + col, poisson)
f1=chisq.test(t(matrix(t2,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t3 ~ row + col, poisson)
f1=chisq.test(t(matrix(t3,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t4 ~ row + col, poisson)
f1=chisq.test(t(matrix(t4,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t5 ~ row + col, poisson)
f1=chisq.test(t(matrix(t5,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t6 ~ row + col, poisson)
f1=chisq.test(t(matrix(t6,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t7 ~ row + col, poisson)
f1=chisq.test(t(matrix(t7,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t8 ~ row + col, poisson)
f1=chisq.test(t(matrix(t8,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t9 ~ row + col, poisson)
f1=chisq.test(t(matrix(t9,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t10 ~ row + col, poisson)
f1=chisq.test(t(matrix(t10,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t11 ~ row + col, poisson)
f1=chisq.test(t(matrix(t11,3,3)),correct=F)
c(f1$stat,fit1$dev)
fit1 = glm(t12 ~ row + col, poisson)
```

```
f1=chisq.test(t(matrix(t12,3,3)),correct=F)
c(f1$stat,fit1$dev)
```

14.2 Three-Factor Tables

14.2.1 Testing $[AC][BC]$

This code does not address the difficult problem of identifying the tables. It merely computes G^2 for the 9 listed tables.

Code 14.2.1.

```
t00=c(1,2,2,0,0,2,3,0)
t01=c(1,2,2,0,1,1,2,1)
t02=c(1,2,2,0,2,0,1,2)
t10=c(2,1,1,1,0,2,3,0)
t11=c(2,1,1,1,1,1,2,1)
t12=c(2,1,1,1,2,0,1,2)
t20=c(3,0,0,2,0,2,3,0)
t21=c(3,0,0,2,1,1,2,1)
t22=c(3,0,0,2,2,0,1,2)
i=c(1,2,1,2,1,2,1,2)
j=c(1,1,2,2,1,1,2,2)
k=c(1,1,1,1,2,2,2,2)
i=factor(i)
j=factor(j)
k=factor(k)
fit=glm(t00 ~ i:k + j:k, poisson)
fit$dev
fit=glm(t01 ~ i:k + j:k, poisson)
fit$dev
fit=glm(t02 ~ i:k + j:k, poisson)
fit$dev
fit=glm(t10 ~ i:k + j:k, poisson)
fit$dev
fit=glm(t11 ~ i:k + j:k, poisson)
fit$dev
fit=glm(t12 ~ i:k + j:k, poisson)
fit$dev
fit=glm(t20 ~ i:k + j:k, poisson)
fit$dev
fit=glm(t21 ~ i:k + j:k, poisson)
fit$dev
fit=glm(t22 ~ i:k + j:k, poisson)
```

```
fit$dev
```

14.2.2 Testing $[B][AC]$

This provides code for G^2s in $[AC][B]$.

I found the following tabulation listing half the tables to be very useful.

Table $(n_{\cdot 11}, n_{221}, n_{222})$	$\prod_{ijk} n_{ijk}!$	$144/\prod_{ijk} n_{ijk}!$	G^2
300	48	3	10.04
301	8	18	3.45
302	16	9	6.22
310	24	6	7.27
311	4	36	0.68
312	8	18	3.45
320	144	1	13.86
321	24	6	7.27
322	48	3	10.04
402	16	9	6.22
411	36	4	8.32
401	24	6	7.27
412	24	6	7.27
502	144	1	13.86

The other 14 tables reverse C1 and C2, have the same probabilities (which are the entries in the third column divided by twice the third column total, i.e., 252) and the same G^2s. The tables are identified by $(n_{\cdot 11}, n_{221}, n_{222})$. Notice that all nine tables obtained from Table 14.1 by choosing one from the left and one from the right have $n_{\cdot 11} = 3$. All nine tables obtained from Table 14.1 by choosing one from the left and one from the right but then reversing the right and left have $n_{\cdot 11} = 2$. The five tables from Table 14.4 have $n_{\cdot 11} = 4, 5$ and reversing right and left give $n_{\cdot 11} = 1, 0$.

There are redundancies in the following computations of G^2 because I wanted to list all 28 of the tables which is not necessary. First are G^2 for tables defined by Table 14.1.

Code 14.2.2.

```
t300=c(1,2,2,0,0,2,3,0)
t301=c(1,2,2,0,1,1,2,1)
t302=c(1,2,2,0,2,0,1,2)
t310=c(2,1,1,1,0,2,3,0)
t311=c(2,1,1,1,1,1,2,1)
t312=c(2,1,1,1,2,0,1,2)
t320=c(3,0,0,2,0,2,3,0)
t321=c(3,0,0,2,1,1,2,1)
```

```
t322=c(3,0,0,2,2,0,1,2)
i=c(1,2,1,2,1,2,1,2)
j=c(1,1,2,2,1,1,2,2)
k=c(1,1,1,1,2,2,2,2)
i=factor(i)
j=factor(j)
k=factor(k)
fit=glm(t300 ~ i:k + j, poisson)
fit$dev
fit=glm(t301 ~ i:k + j, poisson)
fit$dev
fit=glm(t302 ~ i:k + j, poisson)
fit$dev
fit=glm(t310 ~ i:k + j, poisson)
fit$dev
fit=glm(t311 ~ i:k + j, poisson)
fit$dev
fit=glm(t312 ~ i:k + j, poisson)
fit$dev
fit=glm(t320 ~ i:k + j, poisson)
fit$dev
fit=glm(t321 ~ i:k + j, poisson)
fit$dev
fit=glm(t322 ~ i:k + j, poisson)
fit$dev
```

That gave us 9 of the 28 G^2 values

To compute the next 9 G^2s for tables defined by reversing right and left sides of Table 14.1, essentially the same code is used but the new tables need to be listed. However, **you do not need to run this code because the reversed tables give the same G^2s.**

Code 14.2.3.

```
t200=c(0,2,3,0,1,2,2,0)
t210=c(1,1,2,1,1,2,2,0)
t220=c(2,0,1,2,1,2,2,0)
t201=c(0,2,3,0,2,1,1,1)
t211=c(1,1,2,1,2,1,1,1)
t221=c(2,0,1,2,2,1,1,1)
t202=c(0,2,3,0,3,0,0,2)
t212=c(1,1,2,1,3,0,0,2)
t222=c(2,0,1,2,3,0,0,2)

i=c(1,2,1,2,1,2,1,2)
j=c(1,1,2,2,1,1,2,2)
k=c(1,1,1,1,2,2,2,2)
```

```
i=factor(i)
j=factor(j)
k=factor(k)
fit=glm(t200 ~ i:k + j, poisson)
fit$dev
fit=glm(t210 ~ i:k + j, poisson)
fit$dev
fit=glm(t220 ~ i:k + j, poisson)
fit$dev
fit=glm(t201 ~ i:k + j, poisson)
fit$dev
fit=glm(t211 ~ i:k + j, poisson)
fit$dev
fit=glm(t221 ~ i:k + j, poisson)
fit$dev
fit=glm(t202 ~ i:k + j, poisson)
fit$dev
fit=glm(t212 ~ i:k + j, poisson)
fit$dev
fit=glm(t222 ~ i:k + j, poisson)
fit$dev
```

The final 10 G^2s require entering the five tables defined by Table 14.2.4 and the 5 tables that reverse the right and left sides of those.

Code 14.2.4.

```
# tables from Table 16.4
t402=c(2,2,1,0,1,0,2,2)
t401=c(2,2,1,0,0,1,3,1)
t411=c(3,1,0,1,0,1,3,1)
t412=c(3,1,0,1,1,0,2,2)
t502=c(3,2,0,0,0,0,3,2)
# reverse sides of Table 16.4
# G^2 same as first 5
t120=c(1,0,2,2,2,2,1,0)
t110=c(0,1,3,1,2,2,1,0)
t111=c(0,1,3,1,3,1,0,1)
t121=c(1,0,2,2,3,1,0,1)
t020=c(0,0,3,2,3,2,0,0)
i=c(1,2,1,2,1,2,1,2)
j=c(1,1,2,2,1,1,2,2)
k=c(1,1,1,1,2,2,2,2)
i=factor(i)
j=factor(j)
k=factor(k)
fit=glm(t402 ~ i:k + j, poisson)
```

```
fit$dev
fit=glm(t401 ~ i:k + j, poisson)
fit$dev
fit=glm(t411 ~ i:k + j, poisson)
fit$dev
fit=glm(t412 ~ i:k + j, poisson)
fit$dev
fit=glm(t502 ~ i:k + j, poisson)
fit$dev
fit=glm(t120 ~ i:k + j, poisson)
fit$dev
fit=glm(t110 ~ i:k + j, poisson)
fit$dev
fit=glm(t111 ~ i:k + j, poisson)
fit$dev
fit=glm(t121 ~ i:k + j, poisson)
fit$dev
fit=glm(t020 ~ i:k + j, poisson)
fit$dev
```

To run `fisher.test` on these, I need to rearrange how the table is entered. Recall that earlier we had `t320=c(3,0,0,2,0,2,3,0)`, which was written in as rows within layers. Now, writing the table by columns,

Code 14.2.5.

```
T320=matrix(c(3,0,0,2,0,3,2,0),2)
T320
fisher.test(T320)
```

I chose to run the table 320 because it is one of 4 tables that gives the highest G^2 value. Thus I could compare my probability of seeing this G^2 to `fisher.exact`'s P value to see if they agree. (The first time I did it they did not which lead me to find some mistakes in algebra. I had actually been more concerned that I might have missed some of the possible tables, but my corrected algebra corresponded to the program, so I am confident I found all the tables.)

14.3 General Theory

The R program `exactLoglinTest` gave approximate exact P values for quite general log-linear models by sampling from the allowable tables but it has been removed from the CRAN repository from June 20, 2022 because reported problems were not corrected. The program `clogit` from the `survival` package does exact logistic regression.

I am not aware of any R packages or programs that enumerate all of the appropriate tables for general log-linear models. StatXact and LogXact are not R packages but they enumerate all the appropriate tables for many interesting special cases until the tables get too large for that to be practical.

14.4 Model Testing

The exact inference aspects of the program LogXact seems to focus on model comparisons that involve dropping one parameter out of a model. Although the program allows fitting factor terms, it seems to require the specific parameterization that the group with the largest index has its parameter set to 0.

The work of identifying tables for the example was done without a computer. This discussion is only about finding the G^2s.

Earlier I indexed the tables by $(n_{\cdot 11}, n_{221}, n_{222})$. Every table with the same value of $n_{\cdot 11}$ has the same marginal table $n_{\cdot jk}$ because the 2×2 marginal table has fixed margins $n_{\cdot j\cdot}$ and $n_{\cdot\cdot k}$, so the entry $n_{\cdot 11}$ determines the entire $n_{\cdot jk}$ table. (This would get much harder if $n_{\cdot jk}$ was not 2×2.) To illustrate that all tables with a fixed $n_{\cdot 11}$ have the same G^2, I have done the computation on two such tables.

Code 14.4..

```
rm(list = ls())
i=c(1,2,1,2,1,2,1,2)
j=c(1,1,2,2,1,1,2,2)
k=c(1,1,1,1,2,2,2,2)
i=factor(i)
j=factor(j)
k=factor(k)

t300=c(1,2,2,0,0,2,3,0)
t322=c(3,0,0,2,2,0,1,2)

fit=glm(t300 ~ i:k + j, poisson)
fit1=glm(t300 ~ i:k + j:k, poisson)
fit$dev-fit1$dev

fit=glm(t322 ~ i:k + j, poisson)
fit1=glm(t322 ~ i:k + j:k, poisson)
fit$dev-fit1$dev

t201=c(0,2,3,0,2,1,1,1)
t211=c(1,1,2,1,2,1,1,1)

fit=glm(t201 ~ i:k + j, poisson)
```

```
fit1=glm(t201 ~ i:k + j:k, poisson)
fit$dev-fit1$dev

fit=glm(t211 ~ i:k + j, poisson)
fit1=glm(t211 ~ i:k + j:k, poisson)
fit$dev-fit1$dev

t402=c(2,2,1,0,1,0,2,2)
t412=c(3,1,0,1,1,0,2,2)
t502=c(3,2,0,0,0,0,3,2)

fit=glm(t402 ~ i:k + j, poisson)
fit1=glm(t402 ~ i:k + j:k, poisson)
fit$dev-fit1$dev

fit=glm(t412 ~ i:k + j, poisson)
fit1=glm(t412 ~ i:k + j:k, poisson)
fit$dev-fit1$dev

fit=glm(t502 ~ i:k + j, poisson)
fit1=glm(t502 ~ i:k + j:k, poisson)
fit$dev-fit1$dev

t110=c(0,1,3,1,2,2,1,0)
t111=c(0,1,3,1,3,1,0,1)
t020=c(0,0,3,2,3,2,0,0)

fit=glm(t110 ~ i:k + j, poisson)
fit1=glm(t110 ~ i:k + j:k, poisson)
fit$dev-fit1$dev

fit=glm(t111 ~ i:k + j, poisson)
fit1=glm(t111 ~ i:k + j:k, poisson)
fit$dev-fit1$dev

fit=glm(t020 ~ i:k + j, poisson)
fit1=glm(t020 ~ i:k + j:k, poisson)
fit$dev-fit1$dev
```

The huge issue is finding all of the tables that give a fixed set of sufficient statistics.

14.5 Notes

`fisher.test` implements the network algorithm developed by Mehta and Patel (1983, 1986) and improved by Clarkson, Fan and Joe (1993). Two-sided tests are based on the probabilities of the tables, and take as 'more extreme' all tables with probabilities less than or equal to that of the observed table, the *P*-value being the sum of such probabilities. Agresti, Mehta, and Patel (1990) discuss exact tests for ordered categories.

Chapter 15
Correspondence Analysis

15.1 Introduction

15.2 Singular Value Decomposition Plot

See next section

15.3 Correspondence Analysis Plot

R code for both the SVD and CA plots.

Code 15.3.1.

```
lazer <- read.table(
url("http://stat.unm.edu/~fletcher/LLM/DATA/TAB15-1.DAT"),
# "C:\\E-drive\\Books\\ANREG2\\newdata\\tab5-6.dat",
#"C:\\E-drive\\Books\\LOGLIN3\\DATA\\TAB15-1.dat",
  sep="",col.names=c("O","Rel","Occ"))
attach(lazer)
lazer
laz <- xtabs(O~Rel+Occ)
laz

# fit indep/homogen model to table
fit <- chisq.test(laz,correct=FALSE)
fit
fit$expected
fit$residual

# Singular Value Decomp of sqrt(n) x residual matrix
```

```
caa=svd(fit$residual)
L=diag(caa$d)
L
U=caa$u
U
V=caa$v
V
#This should reproduce fit$residual
PR=U%*%L%*%t(V)
PR

# Pearson Chi-square
X2=PR%*%t(PR)
sum(diag(X2))

#Look at how closely 2-dim SVD reproduces residual matrix
PRca=U[,1:2]%*%L[1:2,1:2]%*%t(V[,1:2])
PRca

#Figure 1
par(mfrow=c(2,1))
#SVD Plot
Ut=U%*%sqrt(L)
Vt=V%*%sqrt(L)
# If done correctly the next line should be
# the residual matrix.
Ut%*%t(Vt)
plot(Ut[,1],Ut[,2],
    xlim=c(-2.5,2.5),ylim=c(-2.5,2.5),
     xlab="Factor 1",ylab="Factor 2",main="SVD Plot")
text(Ut[,1],Ut[,2]-.4,labels=c("Protest","RomCath","Jewish"))
lines(Vt[,1],Vt[,2],pch=15,type="p")
text(Vt[,1]-.125,Vt[,2],labels=c("A","B","C","D"))
text(0,0,labels=c("+"))

#Correspondence Analysis Plot
Pr=rowSums(laz)/sum(laz)
Pc=colSums(laz)/sum(laz)
Dr=diag(1/sqrt(Pr))
Dc=diag(1/sqrt(Pc))
Fr=Dr%*%U%*%L/sqrt(sum(laz))
Fc=Dc%*%V%*%L/sqrt(sum(laz))
plot(Fr[,1],Fr[,2],
    xlim=c(-.7,.3),ylim=c(-.3,.3),
     xlab="Factor 1",ylab="Factor 2",main="CA Plot")
```

```
text(Fr[,1],Fr[,2]-.06,labels=c("Protest","RomCath","Jewish"))
lines(Fc[,1],Fc[,2],pch=15,type="p")
text(Fc[,1]-.03,Fc[,2],labels=c("A","B","C","D"))
text(0,0,labels=c("+"))
par(mfrow=c(1,1))


# Figure 2
#SVD Plot
Ut=U%*%sqrt(L)
Vt=V%*%sqrt(L)
# If done correctly the next line should be
# the residual matrix.
Ut%*%t(Vt)
plot(Ut[,1],Ut[,2],
    xlim=c(-2.5,2.5),ylim=c(-2.5,2.5),
     xlab="Factor 1",ylab="Factor 2",main="SVD Plot")
text(Ut[,1],Ut[,2]-.15,labels=c("Protest","RomCath","Jewish"))
lines(Vt[,1],Vt[,2],pch=15,type="p")
text(Vt[,1]-.15,Vt[,2],labels=c("A","B","C","D"))
text(0,0,labels=c("+"))

# Figure 3 is from ANREG2.

oc=c("A","B","C","D")
i=seq(1,4)
P=c(.185,.244,.224,.347)
RC=c(.157,.216,.196,.431)
J=c(.252,.420,.210,.119)
plot(i,P,type="b",xlab="Occupation",ylab="Proportion",at=F,ylim=c(0.10,0.47))
lines(i,RC,type="b",lty=2)
lines(i,J,type="b",lty=3)
axis(side=1,at=i,labels=oc)
axis(side=2)
legend("topleft",c("Protest","RomCath","Jewish"),lty=c(1,2,3))


# Figure 4
plot(Fr[,1],Fr[,2],
    xlim=c(-.575,.23),ylim=c(-.575,.23),
     xlab="Factor 1",ylab="Factor 2",main="CA Rows Plot")
text(Fr[,1],Fr[,2]-.03,labels=c("Protest","RomCath","Jewish"))
text(0,0,labels=c("+"))

#Figure 5
```

```
plot(Fc[,1],Fc[,2],
    xlim=c(-.25,.25),ylim=c(-.25,.25),
     xlab="Factor 1",ylab="Factor 2",main="CA Columns Plot")
text(Fc[,1]-.015,Fc[,2],labels=c("A","B","C","D"))
text(0,0,labels=c("+"))

#Figure 6
plot(Fr[,1],Fr[,2],
      xlim=c(-.575,.23),ylim=c(-.575,.23),
     xlab="Factor 1",ylab="Factor 2",main="CA Plot")
text(Fr[,1],Fr[,2]-.03,labels=c("Protest","RomCath","Jewish"))
lines(Fc[,1],Fc[,2],pch=15,type="p")
text(Fc[,1]-.03,Fc[,2],labels=c("A","B","C","D"))

# Or you could just use the ca package
#install("ca")
library(ca)
fitt=ca(laz)
plot(fitt)
par(mfrow=c(1,1))
```

15.3.1 Nobel Prize Winners

An example on Nobel prize winners by G8 countries from 1901 to 2015 from a Youtube video by Francois Husson, www.youtube.com/watch?v=Z5Lo1hvZ9fA **Code is incomplete or wrong.** I just liked the data.

Code 15.3.2.

```
rm(list = ls())
nobel = matrix(c(
 4,  3,  2,  4,  1,  4,
 8,  3, 11, 12, 10,  9,
24,  1,  8, 18,  5, 24,
 1,  1,  6,  5,  1,  5,
 6,  0,  2,  3,  1, 11,
 4,  3,  5,  2,  3, 10,
23,  6,  7, 26, 11, 20,
51, 43,  8, 70, 19, 66),6,8)
nobel=t(nobel)
nobel
rowSums(nobel)
colSums(nobel)
# rownames Canada France Germany Italy Japan Russia UK USA
```

```
# colnames Chem Econ Lit Med Peace Physics

# fit indep/homogen model to table
fit <- chisq.test(nobel,correct=FALSE)
fit
fit$expected
fit$residual

# Singular Value Decomp of residual matrix
ca=svd(fit$residual)
L=diag(ca$d)
L
U=ca$u %*% sqrt(L)
U
V=ca$v %*% sqrt(L)
V
#This should reproduce fit$residual
PR=U%*%L%*%t(V)
PR

# Pearson Chi-square
X2=PR%*%t(PR)
sum(diag(X2))

#SVD Plot
plot(U[,1],U[,2],
    xlim=c(-1,1),ylim=c(-1,1),
     ,xlab="Factor 1",ylab="Factor 2")
text(U[,1],U[,2]-.075,labels=
    c("Canada","France","Germany",
    "Italy","Japan","USSR","UK","USA"))
lines(V[,1],V[,2],pch=15,type="p")
text(V[,1]+.075,V[,2],labels=
    c("Chem","Econ","Lit","Med","Peace","Phys"))
```

15.4 Multiple correspondence analysis

You can check my algebra by running the following code for the example. Heaven knows I spent enough time checking my algebra.

Code 15.4.1.

```
rm(list = ls())
laz=matrix(c(
```

```
1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,
1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,
0,1,1,0,0,0,0,1,1,0,0,0,0,1,0,0,
0,0,0,1,0,0,0,0,0,1,1,0,0,0,1,0,
0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,1),16,7)
laz

# fit indep/homogen model to table
fit <- chisq.test(laz,correct=FALSE)
fit
fit$expected
fit$residual
t(fit$residual)%*%fit$residual
t(laz)%*%laz
XtX=t(laz)%*%laz
fitt = chisq.test(XtX[1:3,4:7],correct=F)
fitt$residual
```

Chapter 16
Polya Trees

No, there will be no chapter on Polya trees in this book. Someone just wanted these plots.

16.0.1 Alas

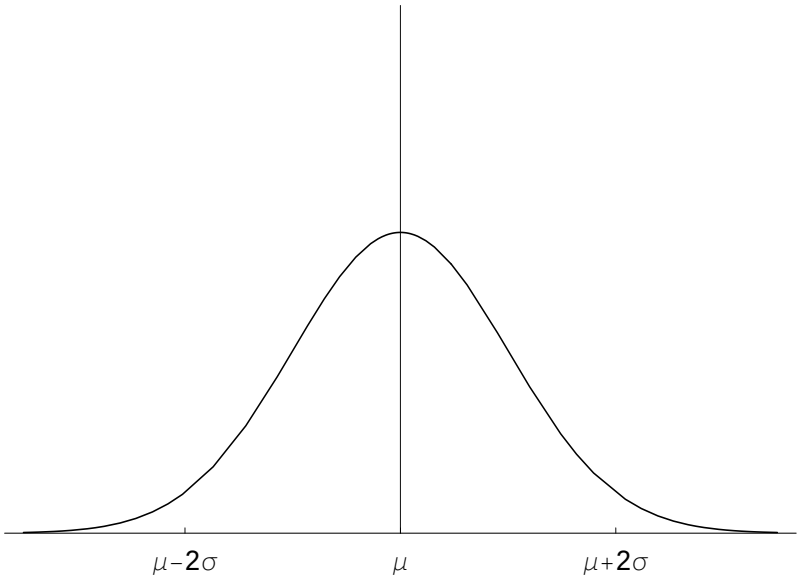


Fig. 16.1 OpenBUGS screenshot.

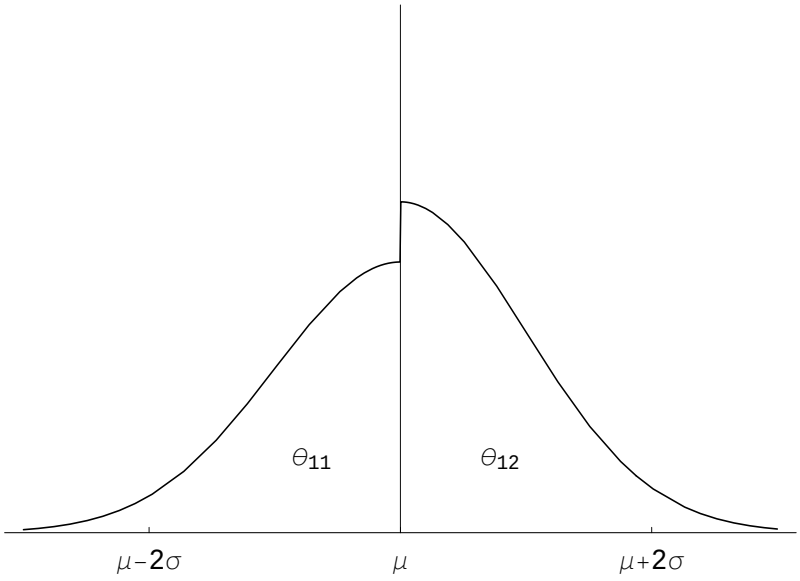


Fig. 16.2 OpenBUGS screenshot.

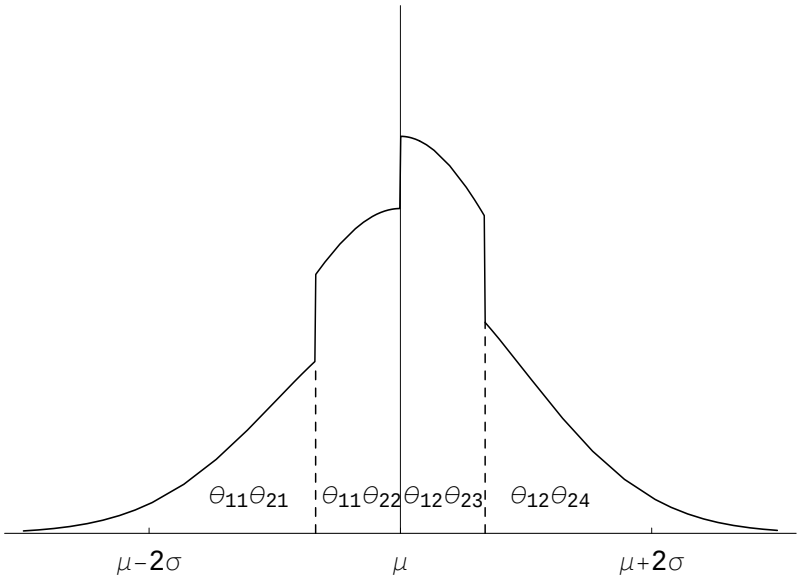


Fig. 16.3 OpenBUGS screenshot.

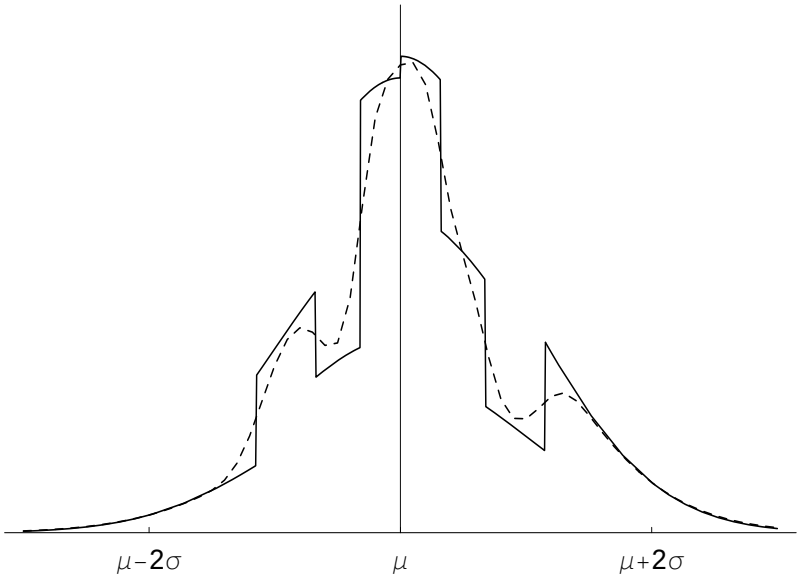
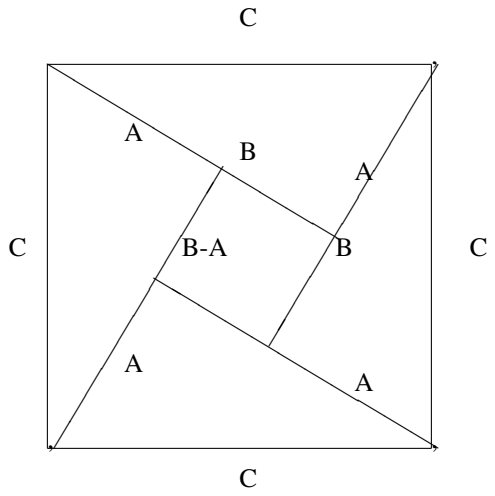


Fig. 16.4 OpenBUGS screenshot.

Chapter 17
Pythagorian Theorem

The outer square has area C^2 which equals the area of the four triangles and the inner square. The inner square has area $(B-A)^2 = B^2+A^2-2AB$ and the 4 triangles have an area of $4\frac{1}{2}AB$, so the total area is A^2+B^2. When I was a graduate student, some undergrad in a computing lab showed this to me.



References

Agresti, Alan, Mehta, C.R., and Patel, N.R. (1990). Exact inference for contingency tables with ordered categories. *Journal of the American Statistical Association*, **85**, 453-458.

Brunswick, A.F. (1971). Adolescent health, sex, and fertility. *American Journal of Public Health*, **61**, 711-720.

Calcagno, Vincent and de Mazancourt, Claire (2010). glmulti: An R Package for Easy Automated Model Selection with (Generalized) Linear Models, *Journal of Statistical Software*, **34(12)**.

Christensen, Ronald (2015). *Analysis of Variance, Design, and Regression: Linear Modeling for Unbalanced Data*, Second Edition. Boca Raton, FL: Chapman and Hall.

Christensen, Ronald (2019). *Advanced Linear Modeling: Statistical Learning and Dependent Data*, Third Edition. New York: Springer-Verlag.

Christensen, Ronald (2020). *Plane Answers to Complex Questions: The Theory of Linear Models*, Fifth Edition. New York: Springer-Verlag.

Christensen, R.; Johnson, W.; Branscum, A.; and Hanson, T. E. (2010). *Bayesian Ideas and Data Analysis: An Introduction for Scientists and Statisticians*. Chapman and Hall/CRC Press, Boca Raton, FL.

Chuang, Christy (1983). Multiplicative-interaction logit models for $I \times J \times 2$ three-way tables. *Communications in Statistics, Theory and Methods*, **12**, 2871-2885.

Clarkson, D. B., Fan, Y. and Joe, H. (1993) A Remark on Algorithm 643: FEXACT: An Algorithm for Performing Fisher's Exact Test in $r \times c$ Contingency Tables. *ACM Transactions on Mathematical Software*, **19**, 484-488.

Geman, S. and Geman, D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6 (6)**, 721-741.

Irwin, J.O. (1949). A note on the subdivision of χ^2 into components. *Biometrika*, **36**, 130-134.

Lancaster, H.O. (1949). The derivation and partition of χ^2 in certain discrete distributions. *Biometrika*, **36**, 117-129.

Lunn, David; Jackson, Christopher; Best, Nicky; Thomas, Andrew; and Spiegelhalter, David (2013). The BUGS Book: A Practical Introduction to Bayesian Analysis. CRC Press: Boca Raton.

Mehta, C. R. and Patel, N. R. (1986). Algorithm 643: FEXACT, a FORTRAN subroutine for Fisher's exact test on unordered $r \times c$ contingency tables. *ACM Transactions on Mathematical Software*, **12**, 154-161.

Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, **21**, 1087-1092.

Tierney, L. (1994). Markov Chains for Exploring Posterior Distributions. *Annals of Statistics*, **22**, 1701-1728. (A good source on the underlying theory of McMC.)

Index

$A - q$ vs AIC, 16
$\sim$, 71
ANREG-II, vii, 14
BIDA, 69
^, vii, 41
~, vii

A-q vs AIC, 16
AIC vs $A - q$, 16
ANREG-II, vii, 14
attach, 118

BIDA, 69
burn-in, 70

caret
  copying problem, 41
column names in data file, 133
copying commands
  problems, vii, 41

DAP, 126
data frame, 118

Fisher scoring, 13

graphical user interface, 137
GUI, 137

inner product, 103

JAGS, 70, 90, 133

Markov chain Monte Carlo, 69
McMC, 69
MultiBUGS, 70

OpenBUGS, 70
outer product, 109

precision, 71

thinning, 70
tilde, 71
  copying problem, vii

WinBUGS, 70