Splus Manual to Accompany Agresti's *Categorical Data Analysis* (1990)

Laura A. Thompson, 1999©

Table of Contents

Introduction

This Splus manual accompanies Agresti's (1990) *Categorical Data Analysis*. I have included examples of almost all of the major (and some minor) analyses introduced by Agresti. The manual chapters parallel those from Agresti so that, for example, in Chapter 2 I discuss using Splus to conduct analyses from Agresti's Chapter 2. In most cases I use the data provided in the text. There are only one or two occasions where I use data from the problems sections in Agresti. With the exception of an extended example from Chapter 4, discussion of results of analyses is limited since the discussion appears in the text. As such, this manual is really not self-contained. It will be helpful (and many times essential) to use the text too to understand completely the analyses in this manual.

In the manual I frequently employ functions that come from user-contributed libraries of Splus. In the text, I note when a particular library is used. These libraries are not included with the Splus software, but can be easily obtained from the internet. I mention in the next section how to get these libraries and how to install them for use with Splus. Each library has its own help manual. I will demonstrate how to access these help files from inside Splus.

The version of Splus I used to do all analyses is 4.5 for Windows. However, Windows versions as far back as 3.0 will do many of the analyses. I do not rely on menus at all. I am also fairly certain that Splus 2000 for Windows will handle most, if not all, of the code in this manual.

To separate Splus input and output from other text in this manual, I have put normal text in Arial font and Splus commands and output in `courier` font. The input is in **`bold`** font, whereas the output is not.

Finally, this manual assumes some familiarity in using the basic commands of Splus. To keep the manual from being too long I do not discuss functions that I use which are not directly related to categorical data analysis. In most cases, typing out the command as it appears in the text can tell you exactly what it does.

A. Libraries in Splus

The libraries used in this manual that do not come with the software are

MASS (B. Ripley) - (used throughout)
Multiv (F. Murtagh) - (used for correspondence analysis)
Design (F. Harrell) - (used for analyses in chapter 9)
Hmisc (F. Harrell) - (support for Design library)
nnet (B. Ripley) - for the function multinom (multinomial logit models)
nolr (M. Mathieson) - (nonlinear ordinal models - supplement to chapter 9)
rmtools (A. Azzalini & M. Chiogna) - (used for chapter 11)
yags (V. Carey) - (used for chapter 11)

All libraries used can be obtained in .zip form from URL http://lib.stat.cmu.edu/DOS/S/Swin or http://lib.stat.cmu.edu/DOS/S. To install a library, first download it to any folder on your computer. Next, "unzip" the file using an "unzipping" program. This will extract all the files into a new folder in the directory into which you downloaded the zip file. (I used unzip95, which is available from the URL above. Winzip should work also. I don't know about pkunzip). Move the entire folder to the library directory under your Splus directory (e.g., c:\Program files\Splus45\library).

To load a library, you can either pull down the File menu in Splus and select Load Library or type one of the following in a script or command window

```
library("libraryname",first=T)
library(libraryname)
```

To use the library's help manual from inside Splus type in a script or command window

help(library="libraryname")

B. Setting contrast types using Options()

The `options` function can be used to set the type of contrasts used in estimating models. The default is Helmert contrasts for factors and polynomial contrasts for ordered factors. I use the default for most estimates, which makes them sometimes different from Agresti's. However, whenever estimates are emphasized, I note how to get his estimates from mine. Whenever I have not used the default setting, I have indicated this prior to obtaining the estimates.

One can find out exactly what the contrasts are in a `glm` type fit by using the functions `model.matrix` and `contrasts`. Issuing the comand `contrasts(model.matrix(fit))` gives the contrasts.

C. Credit for functions

The author of a function is named if the function was not written by me. Whenever I use functions that do not come with Splus, I will mention the Splus library that includes them. The authors for all libraries are mentioned above as well as at the URL address.

D. A note about using Splus Menus

Many of the more common methods I will illustrate can be accomplished via the Splus menus. There are at least two reasons not to use menus for most work (with the exception of graphing). First, using "code" helps one to learn the language. At times, one desires to modify an existing method. Modifications can't be done using the existing menus (albeit, one can write their own menus which *do* contain the modified method). Also, Splus (as well as any interpreted language) is one of the best vehicles to try out new methods without getting bogged down in compiliation errors.

In addition, knowing the code helps in understanding what to put in the slots in the menus. If you want to know what code corresponds to a particular menu command, issue the menu command and call up the History window (using the Window menu). All commands you have issued from menus will be there in (MENU) code form which can be used in a command window or script.

E. Notice of errors

All code has been tested once, but I did not have time to check anything twice. So, please notify me of any errors in the manual or of easier ways to perform tasks. My email address is thompsonL@baylor.edu.

F. References

Agresti, A. (1990). *Categorical Data Analysis*. Wiley

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Cambridge University Press

Chambers, J. & Hastie, T. (1992). *Statistical Models in S*. Chapman & Hall

Harrell, F. (1998). *Predicting Outcomes: Applied Survival Analysis and Logistic Regression*

Ripley B. (1998). On-line complements to *Modern Applied Statistics with SPLUS* (http://www.stats.ox.ac.uk/pub/MASS2)

Selvin, S. (1998). *Modern Applied Biostatistical Methods Using SPLUS*. Oxford University Press

Venables, W. & Ripley, B. (1997). *Modern Applied Statistics with SPLUS*. Springer-Verlag

Wickens, C. (1989). *Multiway Contingency Tables Analysis for the Social Sciences*. LEA

Chapter 2 - Agresti

A. Comparing proportion parameters for two samples, and associated sample statistics

To set up Table 2.3 (heart attack v. no heart attack) on p. 17, in Agresti, type:

```
x<-c(104,189) # aspirin, placebo
n<-c(11037,11034)
```

Then, to test H_0:p_1=p_2 (equal probabilities of heart attack), one can use the `prop.test` function.

```
prop.test(x,n,p=(.5,.5))

   2-sample test for equality of proportions with continuity correction

data:  x out of n
X-square = 24.4291, df = 1, p-value = 0

alternative hypothesis: two.sided
95 percent confidence interval:
 -0.010814914 -0.004597134

sample estimates:
 prop'n in Group 1 prop'n in Group 2
        0.00942285        0.01712887
```

We can obtain the p-value by extracting it from `prop.test`:

```
prop.test(x,n)$p.value
[1] 7.709708e-007
```

A one-sided test of the hypotheses, H_0:p_1=p_2 v. H_1:p_1<p_2, can be obtained using the `alt` option:

```
prop.test(x,n,alt="less")$p.value
[1] 3.854854e-007
```

The sample difference of proportions is extracted as `estimate`:

```
temp<-prop.test(x,n)
names(temp$estimate)<-NULL  # optional
temp$estimate[[1]]-temp$estimate[[2]]
[1] -0.007706024
```

Other useful quantities are easily computed. Here, I calculate the relative risk and odds ratio:

Relative risk:

```
temp$estimate[[2]]/temp$estimate[[1]]
[1] 1.817802
```

Odds ratio:

```
x[2]*(n[1]-x[1])/(x[1]*(n[2]-x[2]))
[1] 1.832054
```

B. Creating Cross-Classified Tables

One can create a cross-classified table out of Table 2.4 (job satisfaction) p. 21, using the following commands:

```
income<-c("<6000","upto15000","upto25000",">25000")
jobsat<-c("very dissat","little","moderate","very sat")
datalabel<-list(income,jobsat)
table.2.4<-fac.design(c(4,4),factor.names=datalabel) # sets up the combinations of
   the levels as a factorial design, using labels datalabel
data<-c(20,22,13,7,24,38,28,18,80,104,81,54,82,125,113,92)
table.2.4<-cbind(table.2.4,data) # add the data
dimnames(table.2.4)[[2]]<-c("income","jobsat","count") # change names of table.2.4
crosstabs(count~income+jobsat,table.2.4)
```

```
Call:
crosstabs(count ~ income + jobsat, table.2.4)
901 cases in table
+----------+
|N         |
|N/RowTotal|
|N/ColTotal|
|N/Total   |
+----------+
income |jobsat
       |vrydsst|little |moderat|verysat|RowTotl|
-------+-------+-------+-------+-------+-------+
<6000  | 20    | 24    | 80    | 82    |206    |
       |0.0971 |0.1165 |0.3883 |0.3981 |0.23   |
       |0.3226 |0.2222 |0.2508 |0.1990 |       |
       |0.0222 |0.0266 |0.0888 |0.0910 |       |
-------+-------+-------+-------+-------+-------+
up15000| 22    | 38    |104    |125    |289    |
       |0.0761 |0.1315 |0.3599 |0.4325 |0.32   |
       |0.3548 |0.3519 |0.3260 |0.3034 |       |
       |0.0244 |0.0422 |0.1154 |0.1387 |       |
-------+-------+-------+-------+-------+-------+
up25000| 13    | 28    | 81    |113    |235    |
       |0.0553 |0.1191 |0.3447 |0.4809 |0.26   |
       |0.2097 |0.2593 |0.2539 |0.2743 |       |
       |0.0144 |0.0311 |0.0899 |0.1254 |       |
-------+-------+-------+-------+-------+-------+
>25000 |  7    | 18    | 54    | 92    |171    |
       |0.0409 |0.1053 |0.3158 |0.5380 |0.19   |
       |0.1129 |0.1667 |0.1693 |0.2233 |       |
       |0.0078 |0.0200 |0.0599 |0.1021 |       |
-------+-------+-------+-------+-------+-------+
ColTotl|62     |108    |319    |412    |901    |
       |0.069  |0.120  |0.354  |0.457  |       |
-------+-------+-------+-------+-------+-------+
Test for independence of all factors
   Chi^2 = 11.98857 d.f.= 9 (p=0.2139542)
   Yates' correction not used
```

See the Splus manual for more uses of crosstabs.

C. Measures of Association - Gamma

Here is a function for computing Goodman and Kruskal's gamma:

```
Gamma.f<-function(x,verbose=F)
{
   # x is a matrix of counts
   n<-nrow(x)
   m<-ncol(x)
   res<-numeric((n-1)*(m-1))
   for(i in 1:(n-1)) { for(j in 1:(m-1)) res[j+(m-1)*(i-1)]<-
   x[i,j]*sum(x[(i+1):n,(j+1):m]) }
   C<-sum(res)
   res<-numeric((n-1)*(m-1))
   iter<-0
   for(i in 1:(n-1)) for(j in 2:m) { iter<-iter+1; res[iter]<-
   x[i,j]*sum(x[(i+1):n,1:(j-1)]) }
   D<-sum(res)
   gamma<-(C-D)/(C+D)
   if(verbose) return( gamma=gamma, C=C, D=D)
      else return(gamma)
   }
```

Using it on table 2.4, we get

```
Gamma.f(matrix(table.2.4[,3],byrow=F,ncol=4) ,verbose=T)
$gamma:
[1] 0.1265461

$C:
[1] 109520

$D:
[1] 84915
```

Selvin (1998) computes the number of concordant and discordant pairs using the `outer` function along with `ifelse` statements. However, the procedure is very memory intensive and will not work with the default memory allocation (this is changed using `options`). The function above takes .05078125 CPU seconds. Selvin's function takes over 30 seconds (on a system with 133 MH, 32 MB RAM and half the hard disk empty).

Other measures of association can be computed immediately from the chisquare value output from `chisq.test` (e.g., phi, Cramer's V, Cramer's C). See Selvin for more details.

Chapter 3 - Agresti

A. Chi-square Tests

This section uses the data in Table 3.2 (p.46-47).

First, I set up the data and the expected counts:

```
pneumonia.counts<-c(30,63,63)
expected<-c(38.1,39,78.9)
```

We can compute Pearson's chisquare statistic and obtain the p-value using the function `.pearson.x2`*:

```
x2value<-unlist(.pearson.x2(pneumonia.counts,expected),use.names=F)
[1] 19.69546

1-pchisq(x2value,df=1)
[1] 9.081684e-006
```

*The function `.pearson.x2` is not well-known. But, you can find it used in `chisq.test`. `chisq.test` does not allow one to set the expected frequencies. With the commands above, I "unlisted" the result of `.pearson.x2` since the result is a list and I wanted a numeric vector.

For calculating GOF tests to named parametric distributions, use `chisq.gof`. For example, taking the data from problem 3.3, we get:

```
m<-(200-109)/200 # lambda
data<-rep(c(0,1,2,3,4),c(109,65,22,3,1)
chisq.gof(data,cut.points=c(-1,0,1,2,3,4),dist="pois",lambda=m)

Warning messages:
Warning in chisq.gof(data, cut.points = c(-1, 0, 1, 2, : Expected counts < 5. Chi-
squared approximation may not be appropriate.

   Chi-square Goodness of Fit Test

data:  data
Chi-square = 12.5698, df = 4, p-value = 0.0136
alternative hypothesis:
  True cdf does not equal the poisson Distn. for at least one sample point.
```

To test independence of two factors, such as those in Table 3.5 (p. 52), I compute Pearson's chi-square statistic. A warning is apt here, as we have fewer than five expected counts for some cells, and the chi-squared approximation requires at least five. An exact test would be more appropriate (see the next section).

```
schizo.mat<-matrix(c(90,12,78,13,1,6,19,13,50),byrow=T,ncol=3)
chisq.test(schizo.mat,correct=F)

Warning messages:
Warning in chisq.test(schizo.mat, correct = F): Expected counts < 5. Chi-square
approximation may not be appropriate.

   Pearson's chi-square test without Yates' continuity correction

data:  schizo.mat
```

```
X-square = 22.3777, df = 4, p-value = 0.0002
```

To compute a likelihood ratio test (LRT) statistic, I modified `.pearson.x2` to obtain the expected frequencies.

```
expected<-my.pearson.x2(schizo.mat)$expected
2*sum(schizo.mat*log(schizo.mat/expected))
[1] 23.03619
```

Specifically, I modified the third-to-last line in `.pearson.x2` to get the expected frequencies:

```
ret.val <- list(X2 = sum((abs(expected - observed) - (if(yates) 0.5 else 0))^2/
   expected), expected = expected)
```

B. Fisher Exact Test

To demonstrate Fisher's Exact Test, Agresti uses the data from Table 3.7 (p.61).

The function `fisher.test` gives a two-sided test:

```
test<-fisher.test(matrix(c(3,1,1,3),byrow=T,ncol=2))

   Fisher's exact test

data:  matrix(c(3, 1, 1, 3), byrow = T, ncol = 2)
p-value = 0.4857
alternative hypothesis: two.sided
```

To get the one-sided p.value, type:

```
test$p.value/2
[1] 0.2428572
```

Note the one-sided p-value can be obtained using `dhyper`

```
sum(dhyper(q=c(3,4),m=4,n=4,k=4))
[1] 0.2428571
```

The next demonstration illustrates how to get the exact conditional test mentioned on p. 64.

Table 3.8

```
temp<-matrix(c(25,25,12,0,1,3),byrow=T,ncol=3)

     [,1] [,2] [,3]
[1,]   25   25   12
[2,]    0    1    3

Gamma.f(temp,T)
$gamma:
[1] 0.8716578

$C:
[1] 175

$D:
[1] 12
```

To get the p-value, I use the functions `factorial` and `prod`:

```
num<-(prod(factorial(rep(1,2)%*%temp))*prod(factorial(rep(1,3)%*%t(temp))))
den<-(factorial(sum(temp))*prod(factorial(temp)))
term1<-(num/den)

temp<-matrix(c(25,26,11,0,0,4),byrow=T,ncol=3) # only other table with C-D as least
   as large

num<-(prod(factorial(rep(1,2)%*%temp))*prod(factorial(rep(1,3)%*%t(temp))))
den<-(factorial(sum(temp))*prod(factorial(temp)))
term2<-(num/den)

term1+term2  # sum the two probabilities
[1] 0.01830808
```

Chapter 4 - Agresti

A. Generalized Linear Models

To illustrate a logit, probit, and linear probability model, Agresti uses Table 4.2 (Cancer remissions, p. 88). To set up the data, type,

```
n<-rep(c(2,3,1,3,2,1,3),c(2,3,1,1,1,5,1))
remissions<-rep(c(0,1,2,1,0,1,0,1,2),c(5,1,1,1,1,2,1,1,1))
LI<-rep(c(8,10,12,14,16,18,20,22,24,26,28,32,34,38),n)

LIlabels<-c("8-12","14-18","20-24","26-32","34-38")
LI.fac<-factor(cut(unique(LI),breaks=c(7,13,19,25,33,39)),labels=LIlabels) # make a
   factor out of LI
temp<-table(LI.fac,remissions) # cross-classify LI.fac and remissions
temp2<-aggregate(cbind(LI.fac,n),by=LI.fac,FUN="sum")["n"] # get table 4.3 from
   table 4.2 (first three columns)
cancer.fr<-cbind.data.frame(LIlabels,temp2,temp[,2]+2*temp[,3],row.names=NULL)
names(cancer.fr)<-c("LI","no.cases","no.remissions")

cancer.fr
```

```
     LI no.cases no.remissions
1  8-12        7             0
2 14-18        7             1
3 20-24        6             3
4 26-32        3             2
5 34-38        4             3
```

To calculate and plot empirical logits, type:

```
attach(cancer.fr)
plot(1:5,log((no.remissions+.5)/(no.cases-no.remissions+.5)),ylab="logit",xlab="")
```

(plot not shown)

Now, I fit a glm. Since we have zeroes as *responses*, I use the function `glim`.

For a logit link:

```
res1<-glim(x=unique(LI), y=remissions,n=n, error="binomial", link="logit",
   resid="deviance")

glim.print(res1)
```

```
             coef se(coef)     z      p  Deviance df change     p
Intercept -3.7771 1.37681  -2.74 0.0061     23.96 13
       X1  0.1449 0.05925   2.44 0.0145     15.66 12 8.3    0.004
```

To obtain the odds ratio, type:

```
as.numeric( exp(res1$coef["X1"]) ) # as.numeric strips the labeling here
```

```
[1] 1.155881
```

To obtain the Wald statistic (p.89), extract the ASE. The Wald statistic is z^2 from the `glim` output above.

```
ASE<-sqrt(res1$var[2,2])

(res1$coef["X1"]/ASE)^2
```

```
       X1
5.978493
```

One can obtain the LR statistic under the column "change" in the glim output above. It can also be obtained using:

```
res1$deviance[1]-res1$deviance[2]
[1] 8.298801
```

Now, I demonstrate the same analysis using a probit link.

```
res2<-glim(x=unique(LI),y=remissions, n=n, error="binomial", link="probit",
   resid="deviance")

glim.print(res2)
             coef se(coef)     z      p  Deviance df change      p
Intercept -2.31710 0.75970  -3.05 0.0023     23.96 13
       X1  0.08782 0.03291   2.67 0.0076     15.44 12 8.5    0.0035
```

And, an identity link (least squares fit):

```
remissions<-c(rep(0,13),1,1,1,0,1,0,0,1,1,0,1,1,1,0) # reset remissions to 0/1
res3<-
   glm(remissions~LI,familiy=binomial(link=identity),control=glm.control(maxit=30))
summary(res3)$coefficients

                  Value  Std. Error   t value
(Intercept) -0.22529619 0.190499859 -1.182658
         LI  0.02782841 0.008630076  3.224584
```

We can obtain fitted probabilities for all three link functions.

Here are the fitted logistic probabilities:

```
logits<-unique(res1$coef["Intercept"] + LI*res1$coef["X1"])
pi.logis<-exp(logits)/(1+exp(logits))
matrix(pi.logis*rep(c(2,3,1,3,2,1,3),c(2,3,1,1,1,5,1)))
 [1,] 0.1359497
 [2,] 0.1775804
 [3,] 0.3455754
 [4,] 0.4445022
 [5,] 0.5657130
 [6,] 0.2369278
 [7,] 0.8796127
 [8,] 0.7132021
 [9,] 0.4254542
[10,] 0.4973257
[11,] 0.5693078
[12,] 0.7023425
[13,] 0.7591824
[14,] 2.5473357
```

probit probabilities:

```
probits<-unique(res2$coef["Intercept"] + LI*res2$coef["X1"])
pi.probits<-pnorm(probits)
matrix(pi.probits,ncol=1)
 [1,] 0.05330968
 [2,] 0.07521720
```

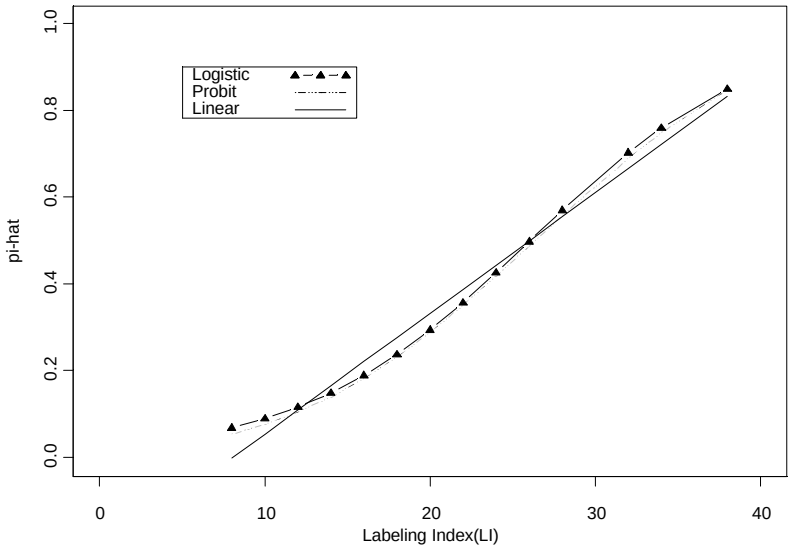
```
 [3,] 0.10339863
 [4,] 0.13855294
 [5,] 0.18107756
 [6,] 0.23096009
 [7,] 0.28770180
 [8,] 0.35029147
 [9,] 0.41724123
[10,] 0.48668632
[11,] 0.55653854
[12,] 0.68911873
[13,] 0.74823039
[14,] 0.84615627
```

linear probabilities:

```
matrix(predict(res3))
 [1,] -0.002668923
 [2,] -0.002668923
 [3,]  0.052987892
 [4,]  0.052987892
 [5,]  0.108644708
 [6,]  0.108644708
 [7,]  0.108644708
 [8,]  0.164301523
 [9,]  0.164301523
[10,]  0.164301523
[11,]  0.219958339
[12,]  0.219958339
[13,]  0.219958339
[14,]  0.275615154
[15,]  0.331271970
[16,]  0.331271970
[17,]  0.331271970
[18,]  0.386928785
[19,]  0.386928785
[20,]  0.442585601
[21,]  0.498242416
[22,]  0.553899232
[23,]  0.665212863
[24,]  0.720869678
[25,]  0.832183309
[26,]  0.832183309
[27,]  0.832183309
```

To reproduce Figure 4.2, use the following commands:

```
LI.plot<-unique(LI)
plot(LI,predict(res3),type="n",xlim=c(0,40),ylim=c(-.005,1.0),xlab="Labeling
   Index(LI)",ylab="pi-hat") # set-up plot using the largest spread of values
lines(LI.plot,pi.logis,type="b",pch=17)
lines(LI.plot,pi.probits,type="l",lty=5)
lines(LI,predict(res3),type="l",lty=1)
key(x=5,y=.9,text=list(c("Logistic","Probit","Linear")),lines=list(type=c("b","l","l
   ")),lty=c(1,5,1),pch=c(17,1,1),divide=3,border=T)
```



B. Logit models for categorical data

For this section, I use the data in Table 4.4 (p.93).

```
BP<-factor(c("<117","117-126","127-136","137-146","147-156","157-166","167-
   186",">186"))
CHD<-c(3,17,12,16,12,8,16,8)
n<-c(156,252,284,271,139,85,99,43)
```

To fit a GLM to Table 4.4, type:

Saturated Model (with null deviance):

```
resCHD<-glm(cbind(CHD,n-CHD)~BP,family=binomial(link=logit))
summary(resCHD)$null.deviance
[1] 30.02257
```

Calculate sample logits:

```
matrix(predict(resCHD),ncol=1)
[1,] -3.931826
[2,] -2.626372
[3,] -3.120895
[4,] -2.768675
[5,] -2.359280
[6,] -2.264364
[7,] -1.646252
[8,] -1.475907
```

Linear logit model

To illustrate fitting a linear logit model, Agresti uses the blood pressure data in Table 4.5 (p. 94).

```
scores<-c(seq(from=111.5,to=161.5,by=10),176.5,191.5)
resLL<-glm(CHD/n~scores,family=binomial,link=logit,weights=n)
summary(resLL)$coefficients
```

```
                 Value  Std. Error   t value
(Intercept) -6.08202789 0.723980447 -8.400818
     scores  0.02433821 0.004841677  5.026814
```

To compute the LR statistic, type:

```
res$null.deviance-res$deviance
[1] 24.11341
```

To calculate estimated probabilities and expected frequencies, use `fitted.values`:

```
probs<-fitted.values(resLL)
matrix(n*probs,ncol=1)
[1,]  5.194869
[2,] 10.606767
[3,] 15.072743
[4,] 18.081622
[5,] 11.616362
[6,]  8.856988
[7,] 14.208763
[8,]  8.361957
```

Here, I reestimate the linear logit model using an ordered factor:

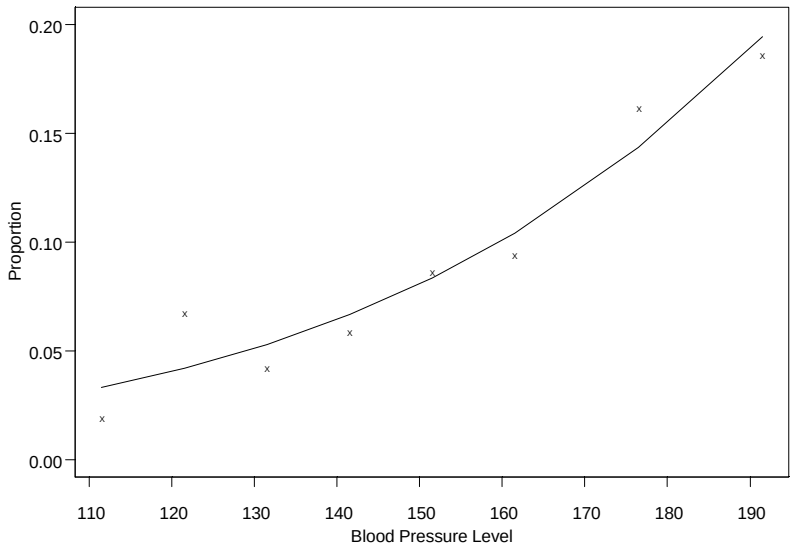
```
BPo<-as.ordered(BP) # make BP an ordered factor
res<-glm(cbind(CHD,n-CHD)~BPo,family=binomial(link=logit))

summary(res)$coefficients
                 Value Std. Error     t value
(Intercept) -2.52419635  0.1258519 -20.0568868
      BPo.L  0.57561274  0.3682757   1.5629943
      BPo.Q  0.04585354  0.3277022   0.1399244
      BPo.C  0.24410276  0.3385445   0.7210359
    BPo ^ 4  1.39413030  0.4080938   3.4162009
    BPo ^ 5  1.01019203  0.3879685   2.6037991
    BPo ^ 6  0.89519565  0.3258823   2.7469910
    BPo ^ 7  0.46217593  0.3256979   1.4190326
```

The default setting for contrasts with ordered factors is polynomial contrasts. With eight levels of BP, we can fit up to septic effects. However, it is likely that any significant effects above cubic are spurious.

To reproduce Figure 4.3, use the following commands:

```
plot(scores,CHD/n,pch="X",yaxt="n",xaxt="n",ylim=c(0,.2),xlab="Blood Pressure
   Level",ylab="Proportion")
axis(side=1,ticks=T,at=seq(from=110,to=200,by=10))
axis(side=2,ticks=T,at=seq(from=0,to=.2,by=.05))
lines(scores,probs,type="l")
```



I will use the data from Table 4.7 (p.106) to mention some properties of glm objects.

To set up the data, type:

```
x<-c(1.691,1.724,1.755,1.784,1.811,1.837,1.861,1.884)
n<-c(59,60,62,56,63,59,62,60)
y<-c(6,13,18,28,52,53,61,60)
```

The following two calls give the same estimates for this problem:

```
res1<-glim(y=y,x=x,n=n,error="binomial",link="loglog")
res1<-glm(y/n~x,weights=n,family=binomial(link=cloglog)) # this is the res1 used in
   the following commands
```

The *summary* for a glm.object has the following attributes that can be extracted:

```
attributes(summary(res1))
$names:
 [1] "call"           "terms"          "coefficients"   "dispersion"
 [5] "df"             "deviance.resid" "cov.unscaled"   "correlation"
 [9] "deviance"       "null.deviance"  "iter"           "nas"

$class:
[1] "summary.glm"
```

For example, to extract the estimated coefficients, type:

```
summary(res1)$coefficients

               Value Std. Error   t value
(Intercept) -39.52232   3.234805 -12.21784
          x  22.01478   1.796516  12.25415
```

The `glm.object` itself has the following components. It inherits all the attributes of `lm.objects`

```
attributes(res1)
$names:
 [1] "coefficients"      "residuals"         "fitted.values"
 [4] "effects"           "R"                 "rank"
 [7] "assign"            "df.residual"       "weights"
```

```
[10] "prior.weights"     "family"            "linear.predictors"
[13] "deviance"          "null.deviance"     "call"
[16] "iter"              "y"                 "contrasts"
[19] "terms"             "formula"           "control"

$class:
[1] "glm" "lm"
```

For example, the fitted values in column 4 in Table 4.7 are

```
n*res1$fitted.values
        1        2        3        4        5        6        7        8
 5.653471 11.28156 20.94184 30.33947 47.68091 54.18769 61.11662 59.94814
```

The same answer can be obtained by the call `n*fitted(res1)`. The functions, `fitted`, `resid`, `coef` are shortened versions to extract fitted values, residuals, and coefficients from glm objects.

The data in Table 4.7 can be fit using the methods discussed above. A complementary log-log link is specified using the link="cloglog" option for function `glim` and the family=binomial(link=cloglog) option for function `glm`.

Model Diagnostics - Residuals

To demonstrate how to obtain the residuals in Agresti's section 4.6.1, I use again the Blood Pressure / Heart Disease data in Table 4.4 to reproduce Table 4.8.

First, fit the independence model:

```
res<-glm(cbind(CHD,n-CHD)~1,family=binomial(link=logit))
n*fitted.values(res)
[1] 10.799097 17.444695 19.659895 18.759970 9.622272  5.884123  6.853273  2.976674
```

Get Pearson residuals for the independence model:

```
matrix(resid(res,type="pearson"))
           [,1]
[1,] -2.4599611
[2,] -0.1103592
[3,] -1.7906464
[4,] -0.6604895
[5,]  0.7945128
[6,]  0.9041221
[7,]  3.6215487
[8,]  3.0178895
```

Now look at the linear logit model:

Get fitted values for the linear logit model:

```
matrix(n*fitted.values(resLL))
[1,]  5.194869
[2,] 10.606767
[3,] 15.072743
[4,] 18.081622
[5,] 11.616362
[6,]  8.856988
[7,] 14.208763
[8,]  8.361957
```

Get Pearson residuals for the linear logit model:

```
matrix(resid(resLL,type="pearson"))
[1,] -0.9782938
[2,]  2.0037958
[3,] -0.8127218
[4,] -0.5064454
[5,]  0.1175384
[6,] -0.3041875
[7,]  0.5134918
[8,] -0.1394930
```

Deviance residuals are obtained using type="deviance". Residuals can be obtained from `glim` fits by extracting them via the dollar sign, $residuals

C. Overdispersed Poisson Model (Negative Binomial)

To estimate a negative binomial model, one must create a family. The function `make.family` can be used to make a new glm family. You must specify the name of the family, the link functions allowed for it (e.g., logit, log, cloglog) and their derivatives, inverses, and initialization expressions (used to initialize the linear predictor for Fisher scoring algorithm), and the variance and deviance functions.

The function `negative.binomial` is available from Venables and Ripley's MASS library available from the website, www.statlib.cmu.edu. In the function listing, you can see where the links and variances lists are defined and where the function `make.family` is used. The reason why the link functions are not actually typed out in function form is because all the links already appear in other glm families in Splus. Their information is stored in the matrix `glm.links`. `negative.binomial` accesses the appropriate link function using `glm.links[,link]`.

A simpler version of `negative.binomial` (called `neg.bin`, with only the log link) is also available from MASS, as well as a function called `glm.nb`. The function `glm.nb` assumes Y~negative binomial with pdf $f_Y(y;\ q,\ m) = \frac{\Gamma(q+y)}{\Gamma(q)y!}\frac{m^y q^q}{(m+q)^{q+y}}$, mean m, and variance $m + m^2/q$, and allows one to estimate theta using ML.

```
############ function negative.binomial (MASS library, V&R) #########

negative.binomial<-
function(theta = stop("theta must be specified"), link = "log")
{
   link <- as.character(substitute(link) )
   misnames <- c("log", "identity", "sqrt", "Log", "Identity", "Sqrt")
   corresp <- c(1, 2, 3, 1, 2, 3)
   lmatch <- pmatch(link, misnames, F)
   if(!lmatch)
      stop(paste("Negative binomial links are currently only",
         "\"log\", \"identity\", or\n\t  \"sqrt\"."))
   link <- misnames[corresp[lmatch]]
   var <- list(name = paste("mu + mu^2", format(round(theta, 4)), sep
       = "/"), variance = substitute(function(mu, th = .theta)
   mu * (1 + mu/th), list(.theta = theta)), deviance = substitute(
      function(mu, y, w, residuals = F, th = .theta)
   {
      devi <- 2 * w * (y * log(pmax(1, y)/mu) - (y + th) * log((y +
         th)/(mu + th)))
      if(residuals)
         sign(y - mu) * sqrt(abs(devi))
      else sum(devi)
   }
   , list(.theta = theta)))
```

```
   make.family("Negative Binomial", glm.links[, link], var)
}

######### function neg.bin (MASS library, V&R) ############
neg.bin<-
function(theta = stop("theta must be given"))
{
   nb.lnk <- list(names = "Log: log(mu)", link = function(mu)
   log(mu), inverse = function(eta)
   exp(eta), deriv = function(mu)
   1/mu, initialize = expression(mu <- y + (y == 0)/6))
   nb.var <- list(names = "mu + mu^2/theta", variance = substitute(
      function(mu, th = .Theta)
   mu * (1 + mu/th), list(.Theta = theta)), deviance = substitute(
      function(mu, y, A, residuals = F, th = .Theta)
   {
      devi <- 2 * A * (y * log(pmax(1, y)/mu) - (y + th) * log((y +
         th)/(mu + th)))
      if(residuals)
         sign(y - mu) * sqrt(abs(devi))
      else sum(devi)
   }
   , list(.Theta = theta)))
   make.family("Negative Binomial", link = nb.lnk, variance = nb.var)
}
```

To briefly illustrate its use, I take an excerpt from an example in V&R (1994), using the data set quine included with MASS (this part probably needs permission before I include it):

Quine data set (in Splus): number days absent from school

```
    Eth Sex Age Lrn Days
  1   A   M  F0  SL    2
  2   A   M  F0  SL   11
  3   A   M  F0  SL   14
  4   A   M  F0  AL    5
  5   A   M  F0  AL    5
  6   A   M  F0  AL   13
...
```

```
fit<-glm.nb(formula = Days ~ Sex + Age %in% Sex + Eth %in% Sex + Lrn %in% Sex +
   Eth:Lrn %in% Sex, data = quine, init.theta = 1.59806882991095, link = log) # %in%
   refers to nesting
```

An estimate of theta is obtained from the fit:

```
c(theta=fit$theta, SE=fit$SE)
    theta        SE
 1.598044 0.2128835
```

D. Extended Example (problem 4.7)

This example illustrates some details in using Splus for a logit model. The analysis is patterned after the section on binomial data in V&R (1994, 1997, p. 230).

First, we set the type of contrast to treatment contrasts for factors.

```
options(contrasts=c("contr.treatment","contr.poly")))
```

Now, we get the data set up:

```
view<-rep(1:7,2)
```

```
view
[1] 1 2 3 4 5 6 7 1 2 3 4 5 6 7

Reagan<-c(1,13,44,155,92,100,18,0,0,2,1,0,2,0)

 race<-factor(rep(c("white","nonwhite"),c(7,7)))
 race
 [1] white    white    white    white    white    white    white    nonwhite
 [9] nonwhite nonwhite nonwhite nonwhite nonwhite nonwhite

 Carter<-c(12,57,71,146,61,41,8,6,16,23,31,8,7,4)

 presvotes<-cbind(Reagan,Carter)
```

Now, fit a binomial glm with interaction:

```
presvotes.lg<-glm(presvotes ~ race*view, family=binomial)
summary(presvotes.lg)

Call: glm(formula = presvotes ~ race * view, family = binomial)
Deviance Residuals:
       Min       1Q    Median        3Q      Max
 -1.443006 -1.07026 -0.492815 0.2732216 1.083453

Coefficients:
                Value Std. Error    t value
(Intercept) -4.4022392  1.4030570 -3.1376054
       race  2.3575997  1.4283371  1.6505906
       view  0.3657521  0.3048505  1.1997751
  race:view  0.1299520  0.3108009  0.4181197

(Dispersion Parameter for Binomial family taken to be 1 )

Null Deviance: 185.1567 on 13 degrees of freedom

Residual Deviance: 12.29655 on 10 degrees of freedom

Number of Fisher Scoring Iterations: 4

Correlation of Coefficients:
          (Intercept)       race       view
     race -0.9823011
     view -0.9441010   0.9273914
race:view  0.9260258  -0.9447185 -0.9808546
```

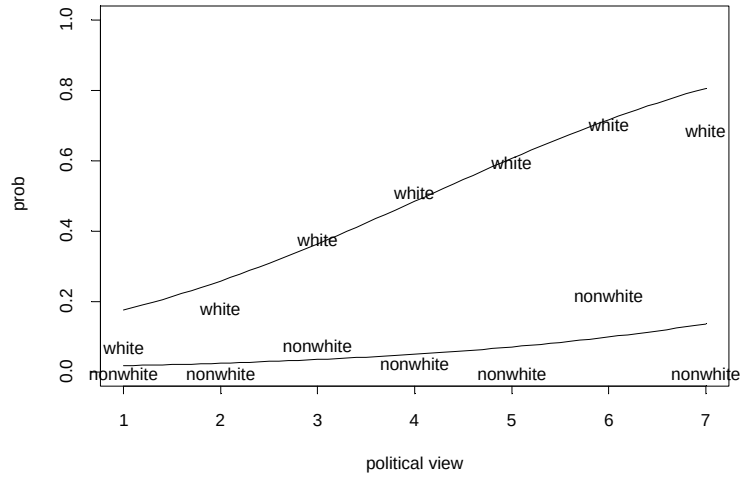
Plot the interaction:

```
 plot(c(1,7),c(0,1), type="n", xlab="political view",ylab="prob")
 text(view,Reagan/(Carter+Reagan),as.character(race))
 ld<-seq(1,7,0.1)
 ld
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7
[19] 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5
[37] 4.6 4.7 4.8 4.9 5.0 5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9 6.0 6.1 6.2 6.3
[55] 6.4 6.5 6.6 6.7 6.8 6.9 7.0
```

Add lines for white and non-white:

```
lines(ld,predict(presvotes.lg,data.frame(view=ld,race=factor(rep("white",
       length(ld)), levels=levels(race))), type="response"))
```

```
lines(ld,predict(presvotes.lg,data.frame(view=ld,race=factor(rep("nonwhite",
      length(ld)), levels=levels(race))),type="response"))
```



We can test for a race difference at a particular view, say view=5. The `I( )` function is used so that (view-5) is interpreted *as is*, meaning as a number, here.

```
presvotes.lgA<-glm(presvotes~race*I(view-5),family=binomial)
summary(presvotes.lgA)
```

```
Call: glm(formula = presvotes ~ race * I(view - 5), family = binomial)
Deviance Residuals:
       Min       1Q    Median        3Q      Max
 -1.443006 -1.07026 -0.492815 0.2732216 1.083453

Coefficients:
                    Value Std. Error    t value
     (Intercept) -2.5734788  0.5037669 -5.1084713
            race  3.0073596  0.5110801  5.8843217
     I(view - 5)  0.3657521  0.3048505  1.1997751
race:I(view - 5)  0.1299520  0.3108009  0.4181197

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 185.1567 on 13 degrees of freedom

Residual Deviance: 12.29655 on 10 degrees of freedom

Number of Fisher Scoring Iterations: 4

Correlation of Coefficients:
                (Intercept)       race I(view - 5)
            race -0.9856907
     I(view - 5)  0.3962650  -0.3905948
race:I(view - 5) -0.3886784   0.4003841 -0.9808546
```

`step` is a function for doing a stepwise analysis. The component `anova` is the most useful. We use it here to find out whether it picks a model with interaction or not. The model with the lowest AIC is considered best.

```
step(presvotes.lg,trace=F)$anova
```

```
Stepwise Model Path
Analysis of Deviance Table
```

```
Initial Model:
presvotes ~ race * view

Final Model:
presvotes ~ race + view


         Step Df  Deviance Resid. Df Resid. Dev      AIC
1                                 10   12.29655 20.29655
2 - race:view  1 0.1737404        11   12.47030 18.47030
```

The independence model Race + view is chosen. Here is the Analysis of deviance:

```
anova(glm(presvotes~race + view, family=binomial),test="Chi")

Analysis of Deviance Table

Binomial model

Response: presvotes

Terms added sequentially (first to last)
     Df Deviance Resid. Df Resid. Dev Pr(Chi)
NULL                    13   185.1567
race  1 95.49444        12    89.6622       0
view  1 77.19194        11    12.4703       0
```

Now, we test a model with parallel lines for each race (common slope), but different intercepts. The term '-1' in the formula removes the common intercept (forces it through 0) so that the coefficient estimate on the 1-coded level of race becomes the new nonzero intercept for that level.

```
presvotes.lgB<-glm(presvotes~race + view - 1, family=binomial)
summary(presvotes.lgB)$coefficients

                  Value Std. Error   t value
racenonwhite -4.9608574 0.53948823 -9.195488
   racewhite -2.0239493 0.26239118 -7.713481
        view  0.4908373 0.05926813  8.281641
```

One result we can obtain from setting different intercepts is a prediction of the viewpoint at which prob(vote for Reagan) = .25, .5, .75 for nonwhites/whites. We can use the function `dose.p` from the MASS library. The second argument to the function (i.e., c(1, 3) or c(2, 3)) refers to the coefficients specifying the common slope and separate intercept. For nonwhites, they are the racenonwhite coefficient and view coefficient. The third argument refers to the probability points (.25, .5, .75).

For nonwhites, we have the following predictions:

```
dose.p(presvotes.lgB,c(1,3),(1:3)/4)
               Dose       SE
p = 0.25:  7.868687 1.029045
p = 0.50: 10.106928 1.160322
p = 0.75: 12.345169 1.334109
```

And, for whites, we have:

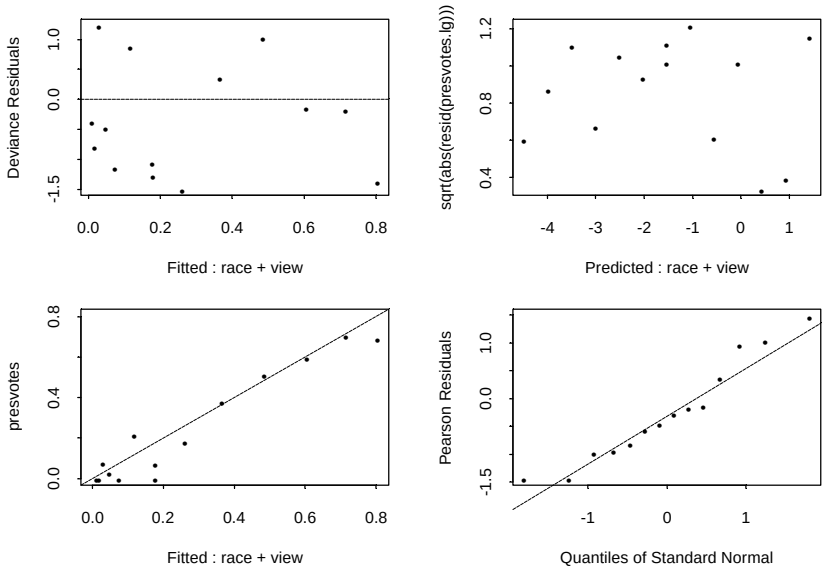
```
dose.p(presvotes.lgB,c(2,3),(1:3)/4)
              Dose        SE
p = 0.25: 1.885221 0.3222993
p = 0.50: 4.123463 0.1499470
p = 0.75: 6.361704 0.2952595
```

To print residual plots for the model: presvotes ~ race + view, type:

```
par(mfrow=c(2,2))
plot(presvotes.lg,ask=T)
```

```
Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Residuals vs Fitted Values
3: plot: Sqrt of abs(Residuals) vs Predictions
4: plot: Response vs Fitted Values
5: plot: Normal QQplot of Std. Residuals
Selection: 1
```



Chapter 5 - Agresti

A. Demonstration of Simpson's Paradox

The set of commands in this section illustrates how to get from Table 5.1 to 5.2 (i.e., collapse a cross-classified table output from `crosstabs`)

To set up the data:

```
DeathP<-c("Yes","No")
Victim.race<-c("White","Black")
Defend.race<-c("White","Black")
table.5.1<-expand.grid(DeathP, Victim.race, Defend.race) # compare this to the
   function fac.design
names(table.5.1)<-c("DeathP","Defendent","Victim")
```

Next, add the counts to `table.5.1` and put both into a data frame:

```
table.5.1<-data.frame(table.5.1,counts=c(19,132,11,52,0,9,6,97))

crosstabs(counts~DeathP+Defendent+Victim,table.5.1, margin=list())
```

(margin=list() causes only the counts to show in the cells – see the help file for more, as well as the default)

```
Victim=White
DeathP |Defendent
       |White  |Black  |RowTotl|
-------+-------+-------+-------+
Yes    | 19    | 11    |30     |
       |       |       |0.140  |
-------+-------+-------+-------+
No     |132    | 52    |184    |
       |       |       |0.860  |
-------+-------+-------+-------+
ColTotl|151    |63     |214    |
       |0.71   |0.29   |       |
-------+-------+-------+-------+

Victim=Black
DeathP |Defendent
       |White  |Black  |RowTotl|
-------+-------+-------+-------+
Yes    |  0    |  6    |6      |
       |       |       |0.054  |
-------+-------+-------+-------+
No     |  9    | 97    |106    |
       |       |       |0.946  |
-------+-------+-------+-------+
ColTotl|9      |103    |112    |
       |0.08   |0.92   |       |
-------+-------+-------+-------+
Test for independence of all factors
   Chi^2 = 122.3975 d.f.= 4 (p=0)
   Yates' correction not used
```

Now, to collapse Table 5.1 to Table 5.2, use

```
crosstabs(counts ~ DeathP + Defendent, table.5.1, margin = list())
```

```
DeathP |Defendent
       |White  |Black  |RowTotl|
-------+-------+-------+-------+
Yes    | 19    | 17    |36     |
       |       |       |0.11   |
-------+-------+-------+-------+
No     |141    |149    |290    |
       |       |       |0.89   |
-------+-------+-------+-------+
ColTotl|160    |166    |326    |
       |0.49   |0.51   |       |
-------+-------+-------+-------+
Test for independence of all factors
   Chi^2 = 0.2214463 d.f.= 1 (p=0.6379401)
   Yates' correction not used
```

Chapter 6 - Agresti

A. Fitting Loglinear Models

If the data are already in a data frame with counts as one variable, as table.5.1 is, the easiest method to use to fit a loglinear model is `glm` with family=poisson and a log link (called a *surrogate poisson*). There is also a function in the MASS library called `loglm` which allows input from a data frame (using a formula). The function `loglin` , which comes with Splus, requires input in the form of an array such as output from `table` or `matrix` or `array`. (`loglm` can also take an array as input). It is easy to get an array from a data frame: issue the `design.table` command with the data frame as the argument. (Note, also, that the function `as.data.frame.array` achieves the opposite effect: that is, returning a data frame as the equivalent expression of an array)

Agresti uses the data in Table 6.3 (Death Penalty Example, p.171-174) to illustrate fitting of a loglinear model.

Since we already have table.5.1 in a data frame, let's use `loglm` from the MASS library. (I will illustrate the other functions later, especially the use of `glm`).

The next set of commands fits the models in table 6.3 (not in the same order)

Model (D,V,P):

```
fitDVP<-loglm(counts~.,data=table.5.1,param=T,fit=T)  # '.' =
   DeathP+Defendent+Victim
```

Obtain expected frequencies:

```
fitted.values(fitDVP)
```

```
, , White
       White    Black
Yes 11.59848 12.03342
 No 93.43220 96.93591

, , Black
        White     Black
Yes  6.070232  6.297866
 No 48.899094 50.732811
```

```
summary(fit)
```

```
Statistics:
                      X^2 df P(> X^2)
Likelihood Ratio 137.9294  4        0
         Pearson 122.3975  4        0
```

Model (VP,DV) :

```
fitVP.DV<-update(fitDVP, .~ DeathP*Victim + Defendent*Victim)
fitted.values(fitVP.DV)
```

```
, , White
        White     Black
Yes  21.16822 0.4821429
 No 129.83177 8.5178576
```

```
, , Black
        White     Black
Yes  8.831776  5.517857
 No 54.168224 97.482140
```

summary(fitVP.DV)

```
Statistics:
                      X^2 df  P(> X^2)
Likelihood Ratio 1.881909  2 0.3902552
         Pearson 1.431339  2 0.4888648
```

Model (DVP) - Saturated:
fit.DVP<-update(fitDVP, .~. ^2 + Defendent*Victim*DeathP)
fitted.values(fit.DVP)

```
, , White
    White Black
Yes    19    11
 No   132    52

, , Black
    White Black
Yes     0     6
 No     9    97
```

summary(fit.DVP)

```
Statistics:
                X^2 df P(> X^2)
Likelihood Ratio   0  0        1
         Pearson  NA  0        1
```

Model (P,DV):
fitP.DV<-loglm(counts~ DeathP + Defendent*Victim,data=table.5.1,param=T,fit=T)

same as
fitP.DV<-update(fitDVP, .~. + Defendent*Victim)

Model (DP,VP,DV) - Note how the formula is written
fitDP.VP.DV<-loglm(counts~(Defendent+Victim+DeathP)^2,data=table.5.1,param=T,fit=T)

same as
`fitDP.VP.DV<-update(fitDVP, .~. ^2)`

B. Partial and Marginal Odds Ratios

Obtaining partial and marginal odds ratios is illustrated in Table 6.4 (p. 173). Here, I show how to reproduce that table.

All 1.0 odds ratios are 1.0 by design. For example, in the independence model, when the victim is white, the odds of a white defendant being given the death penalty are the same as for a black defendant. The non-unity odds ratios must be computed either by using the estimated frequencies (easiest) or by using the parameter estimates. In the latter case, you must derive the expression for the odds ratio, which can be very tedious if the model is complicated. Thus, I have only used one of the models in Table 6.4.

Compute partial and marginal odds ratios for association D-V under model (VP,DV):

Get estimated parameters:

```
lambdaD<-fitVP.DV$param$Defendent
lambdaV<-fitVP.DV$param$Victim
lambdaP<-fitVP.DV$param$DeathP
lambdaDV<-fitVP.DV$param$"Victim X Defendent"
lambdaVP<-fitVP.DV$param$"DeathP X Victim"
```

Partial association (D-V):

```
exp( lambdaDV[1,1]+lambdaDV[2,2]-lambdaDV[1,2]-lambdaDV[2,1] )
[1] 27.43033
```

Marginal association (D-V) - sum over levels of P

```
num<-(exp(lambdaD[1]+lambdaV[1]+lambdaP[1]+lambdaDV[1,1]+lambdaVP[1,1]) +
     exp(lambdaD[1]+lambdaV[1]+lambdaP[2]+lambdaDV[1,1]+lambdaVP[1,2])) *
     (exp(lambdaD[2]+lambdaV[2]+lambdaP[1]+lambdaDV[2,2]+lambdaVP[2,1]) +
     exp(lambdaD[2]+lambdaV[2]+lambdaP[2]+lambdaDV[2,2]+lambdaVP[2,2]))

den<-(exp(lambdaD[2]+lambdaV[1]+lambdaP[1]+lambdaDV[2,1]+lambdaVP[1,1]) +
     exp(lambdaD[2]+lambdaV[1]+lambdaP[2]+lambdaDV[2,1]+lambdaVP[1,2])) *
     (exp(lambdaD[1]+lambdaV[2]+lambdaP[1]+lambdaDV[1,2]+lambdaVP[2,1]) +
     exp(lambdaD[1]+lambdaV[2]+lambdaP[2]+lambdaDV[1,2]+lambdaVP[2,2]))

as.numeric( num/den )
[1] 27.43033
```

Note:I'm pretty certain this calculation is unnecessarily complicated.

C. Comparing nested models using LR tests

Table 6.6 (p.176) in Agresti displays a comparison of the set of hierarchical models (DVP), (DV,P), (VP,DV), (DP,VP,DV) using LR tests. One can reproduce this table using the anova function.

```
anova(fitDVP,fitP.DV,fitVP.DV,fitDP.VP.DV)

Model 1:
 counts  ~  DeathP + Victim + Defendent
Model 2:
 counts  ~  Defendent:Victim + DeathP
Model 3:
 counts  ~  DeathP + Victim + Defendent + DeathP:Victim + Defendent:Victim
Model 4:
 counts  ~  (Defendent + Victim + DeathP)^2

            Deviance df  Delta(Dev) Delta(df) P(> Delta(Dev)
  Model 1 137.9293816  4
  Model 2   8.1316088  3 129.7977728         1        0.00000
  Model 3   1.8819090  2   6.2496997         1        0.01242
  Model 4   0.7007495  1   1.1811596         1        0.27712
Saturated   0.0000000  0   0.7007495         1        0.40253
```

Based on the results, we should keep Model 3, (PV, DV). We cannot reject Model 3 in favor of a simpler model, Model 2 (DV,P).

D. Loglinear Models as Logit Models

To illustrate the relationship between loglinear and logit models, when one factor is treated as the response, I use the data from table 5.1. Here, death penalty is the response.

```
  DeathP Defendent Victim counts
1    Yes     White  White     19
2     No     White  White    132
3    Yes     Black  White     11
4     No     Black  White     52
5    Yes     White  Black      0
6     No     White  Black      9
7    Yes     Black  Black      6
8     No     Black  Black     97
```

```
yes<-table.5.1[DeathP=="Yes",]$counts  # gives (19, 11, 0, 6)
no<-table.5.1[DeathP=="No",]$counts
attach(table.5.1)
```

To fit the logit model:

```
fit<-glm(cbind(no,yes) ~ Victim[c(2,4,6,8)], family=binomial, link="logit")
```

Victim[c(2,4,6,8)] gives (White, White, Black, Black)

```
Coefficients:
 (Intercept) Victim[c(2, 4, 6, 8)]
    2.342709             0.5289706

Degrees of Freedom: 4 Total; 2 Residual
Residual Deviance: 1.881895
```

The relationship between the parameters in the logit model and loglinear model (VP,DV) is displayed here. Note the value for the coefficient on Victim in the logit model:

```
fitVP.DV$param$"DeathP X Victim"[1,1] - fitVP.DV$param$"DeathP X Victim"[1,2]
[1] 0.5289706
```

E. Asymptotic Standard Errors and Wald tests

Standard errors and Wald tests can be obtained from the summary of a `glm.object` but not (directly) from a `loglin` object (or `loglm` object). However, one can always extract the model matrix, X, and use the estimated frequencies to compute (6.12) or (6.13) in Agresti.

I will use the Death Penalty data to show part of Table 6.7.

To obtain the model matrix, use the function `model.matrix` with the first argument a `loglm` fit.

```
X<-model.matrix(fitVP.DV,data=table.5.1)
```

```
(Intercept) DeathP Victim Defendent DeathP:Victim Defendent:Victim
1           1     -1     -1        -1             1                1
2           1      1     -1        -1            -1                1
3           1     -1     -1         1             1               -1
4           1      1     -1         1            -1               -1
5           1     -1      1        -1            -1               -1
6           1      1      1        -1             1               -1
7           1     -1      1         1            -1                1
8           1      1      1         1             1                1
```

Get the total number of cases

```
n<-sum(fitVP.DV$frequencies)
```

and the vector of expected frequencies:
```
mhat<-matrix(fitted.values(fitVP.DV),ncol=1)
```

Then, the asymptotic covariance matrix (6.12) is computed as
```
covbeta<-solve(t(X)%*%(diag(as.numeric(mhat)))%*%X) # poisson sampling approximation
```

And, the ASEs in Table 6.7 should be (?)
```
sqrt(diag(covbeta))[5:6]
[1] 0.09070108 0.09496805  # Table 6.7 has .116 and .095 (the first does not match)
```

F. Probability Estimates

Starting with the fitted values from the model (DV,VP) of the Death Penalty data,

```
, , White
        White     Black
Yes  21.16822 0.4821429
 No 129.83177 8.5178576

, , Black
        White     Black
Yes  8.831776  5.517857
 No 54.168224 97.482140
```

we calculate the estimated probabilities of death penalty (Table 6.8) as

```
fit<-fitted.values(fitVP.DV)
den<-apply(fit,3,colSums)
fit[1,1,1]/den[1,1]
fit[1,2,1]/den[2,1]
fit[1,1,2]/den[1,2]
fit[1,2,2]/den[2,2]

[1] 0.1401869
[1] 0.05357143
[1] 0.1401869
[1] 0.05357143
```

Asymptotic standard errors can be computed using equation (6.15)

```
p<-mhat/n
Dpp<-diag(as.numeric(p))-p%*%t(p)
covp<-(Dpp%*%X%*%solve(t(X)%*%Dpp%*%X)%*%t(X)%*%Dpp)/n
sqrt(diag(covp))
```

G. Analyzing survival rates using loglinear models

For this section, Agresti uses the data in Table 6.12 (p. 190) - Heart valve operations.

I will use the function `glm` to fit the model in Section 6.6.1 because I want to include the offset term, exposure.

First, set up the data
```
table.6.12<-expand.grid(c("Aortic","Mitral"),c("<55","55+"))
table.6.12<-data.frame(table.6.12,Deaths=c(4,1,7,9),Exposure=c(1259,2082,1417,1647))
names(table.6.12)[1:2]<-c("Valve","Age")
attach(table.6.12)
```

Add Risks to Table 6.12:

```
table.6.12<-data.frame(table.6.12,Risk=Deaths/Exposure)
```

```
   Valve Age Deaths Exposure         Risk
1 Aortic <55      4     1259 0.0031771247
2 Mitral <55      1     2082 0.0004803074
3 Aortic 55+      7     1417 0.0049400141
4 Mitral 55+      9     1647 0.0054644809
```

Fit a glm with an offset term for log(Exposure):

```
fit<-glm(Deaths~Valve+Age+offset(log(Exposure)),family=poisson,link=log)
```

```
Coefficients:
 (Intercept)      Valve      Age
   -5.866556 -0.1649332 0.610474

Degrees of Freedom: 4 Total; 1 Residual
Residual Deviance: 3.222511
```

To get the chi-squared statistic, use `resid`:

```
sum(resid(fit,type="pearson")^2)
[1] 3.113503
```

Obtain the log odds of risk for older group and its asymptotic ASE (p. 191) using,

```
c(logit=2*coef(fit)["Age"],ASE=2*summary(fit)$coefficients["Age","Std. Error"])
```

```
 logit.Age       ASE
  1.220948 0.5136586
```

To reproduce the fitted values for the glm in Table 6.13 (p. 192), use the following commands.

```
attach(table.6.12)
mhat<-fitted(fit)
exphat<-fitted(fit)/Exposure
temp<-rbind(mhat,exphat)
array(temp,dim=c(2,2,2),dimnames=list(c("Deaths","Risk"),Valve=c("Aortic","Mitral"),
   Age=c("<55","55+")))
```

```
, , <55
            Aortic      Mitral
Deaths 2.284108702 2.715892496
  Risk 0.001814225 0.001304463

, , 55+
            Aortic      Mitral
Deaths 8.715891783 7.284108228
  Risk 0.006150947 0.004422652
```

Now, Agresti compares the first model with a model that omits the Valve factor:

```
fit2<-glm(Deaths~Age+offset(log(Exposure)),family=poisson,link=log)
```

Model comparison test:

```
anova(fit2,fit,test="Chi")

Analysis of Deviance Table

Response: Deaths

                                 Terms Resid. Df Resid. Dev   Test Df  Deviance
1         Age + offset(log(Exposure))         2   3.789717
2 Valve + Age + offset(log(Exposure))         1   3.222511 +Valve  1 0.5672062

    Pr(Chi)
1
2 0.4513712
```

We keep the model Count~Valve + Age + Offset(log(Exposure)).

To test a large set of models in a stepwise manner, one can use the function `step` with a lower and upper model specified. Cp and AIC criteria are given. Only the final model is printed unless trace=T is specified.

```
step(fit, list(lower = formula(fit), upper =  ~ . + Valve*Age), scale=1, trace=T,
   direction="both")
```

The above specifies both forward and backward stepwise selection of terms with the starting model that was obtained from "fit" (in this case, "fit" is the output from the last `glm` we ran). The lower bound model is also the model represented by "fit". The upper bound model includes a valve x age interaction. Since there is only one term separating the starting model from the upper model, a stepwise selection is not really necessary. Examples using `step` are given below.

H. Modeling survival times using loglinear models

Agresti illustrates fitting survival models as loglinear models using the data in Table 6.14.

Since we have an offset, we'll have to use `glm` again to fit the model.

First, form the data set:

```
table.6.14<-expand.grid(Stage=factor(c(1,2,3)), Histology=c("I","II","III"),
   Time=factor(c(0,2,4,6,8,10,12)))
table.6.14<-
   cbind.data.frame(table.6.14,Deaths=c(9,12,42,5,4,28,1,1,19,2,7,26,2,3,19,1,1,11,9
   ,5,12,3,5,10,1,3,7,10,10,10,2,4,5,1,1,6,1,4,5,2,2,0,0,0,3,3,3,4,2,1,3,1,0,3,1,4,1
   ,2,4,2,0,2,3),
Exposure=c(157,134,212,77,71,130,21,22,101,139,110,136,68,63,72,17,18,63,126,96,90,6
   3,58,42,14,14,43,102,86,64,55,42,21,12,10,32,88,66,47,50,35,14,10,8,21,82,59,39,4
   5,32,13,8,8,14,76,51,29,42,28,7,6,6,10) )
```

Now, perform a loglinear analysis using `glm`:

```
fit<-glm(Deaths~Histology+Time+Stage+offset(log(Exposure)),data=table.6.14,
   family=poisson,link=log)

summary(fit)

Coefficients:
                  Value Std. Error      t value
(Intercept) -2.503854618 0.07788986 -32.14609023
 Histology1  0.081221146 0.06097426   1.33205626
 Histology2  0.008773679 0.04708072   0.18635396
```

```
      Time1 -0.063727050 0.07453977  -0.85494027
      Time2 -0.005334666 0.05151831  -0.10354894
      Time3  0.046995855 0.03964064   1.18554721
      Time4 -0.128608735 0.05015534  -2.56420824
      Time5 -0.033246557 0.03910684  -0.85014685
      Time6  0.001248372 0.03432625   0.03636787
     Stage1  0.235006675 0.08721639   2.69452418
     Stage2  0.363101273 0.03846787   9.43907904

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 175.7178 on 62 degrees of freedom

Residual Deviance: 43.92253 on 52 degrees of freedom

Number of Fisher Scoring Iterations: 4
```

We choose a model using `step`.

```
fit2<-glm(Deaths~Time+offset(log(Exposure)),data=table.6.14,family=poisson,link=log)
      # (lower model)
step(fit,scope=list(lower=formula(fit2),upper=~.^2),direction="both",trace=T)$anova
```

Selecting the "anova" component gives the Analysis of Deviance table.

```
Stepwise Model Path
Analysis of Deviance Table

Initial Model:
Deaths ~ Histology + Time + Stage + offset(log(Exposure))

Final Model:
Deaths ~ Time + Stage + offset(log(Exposure))


         Step Df Deviance Resid. Df Resid. Dev      AIC
1                                52   43.92253 65.92253
2 - Histology  2 1.876473        54   45.79901 63.79901
```

Thus, `step` chooses a model without Histology.

Illustration of using function `loglin`

Finally, I will use the data in Table 6.16 to illustrate the Splus function `loglin` -- loglinear analysis using an iterative proportional fitting algorithm. `loglm` actually calls `loglin`. The difference between the two is the way in which each handles structural zeros (`loglm` tries to correct the df, but doesn't always get it right) and that `loglm` can take input in the form of a formula *or* an array. `loglin` requires input in the form of an array, and requires one to specify the marginal totals to be fit. Other differences, as well as further discussion, can be found in the *On-line Complements* to V&R (1997).

Here is the full call to loglin:

```
loglin(table, margin, start = rep(1, length(table)), fit = FALSE, eps = 0.1, iter =
   20, param = FALSE, print = TRUE)
```

Some examples of specifying marginal totals for three variables are:
conditional independence model: margin=list(c(1,2),c(1,3))
independence model: margin=list(1,2,3)

saturated model: margin=list(c(1,2,3))

For table 6.16, first create an array of frequencies:

```
temp<-c("G.Dis","Mid.Pos","G.App")
Att.Abortion<-rep(rep(temp,3),c(209,101,237,151,126,426,16,21,138))
Att.Abortion<-ordered(Att.Abortion,levels=c("G.Dis","Mid.Pos","G.App")) # I ordered
   this so that the table margins are not automatically placed in alphbetical order
Schooling<-rep(ordered(c("LessHS","HS","MoreHS"),levels=c("LessHS","HS","MoreHS")),
   c(209+101+237,151+126+426,16+21+138))
table.6.16<-table(Schooling,Att.Abortion)
```

```
       G.Dis Mid.Pos G.App
LessHS   209     101   237
    HS   151     126   426
MoreHS    16      21   138
```

Here I perform a loglinear analysis testing an independence model. The marginal totals to fit are those for the variables one and two. Thus, margin=list(1, 2). I also request parameter estimates (param=T) and fitted frequencies (fit=T)

```
loglin(table.6.16,margin=list(1,2),param=T,fit=T)
```

```
2 iterations: deviation 0
$lrt:
[1] 96.52666

$pearson:
[1] 93.03375

$df:
[1] 4

$margin:
$margin[[1]]:
[1] 1

$margin[[2]]:
[1] 2


$fit:
          G.Dis   Mid.Pos     G.App
LessHS 144.33122  95.19719 307.47156
    HS 185.49333 122.34666 395.16000
MoreHS  46.17544  30.45614  98.36842

$param:
$param$constant:
[1] 4.78923

$param$"1":
    LessHS        HS     MoreHS
 0.2962515 0.5471598 -0.8434113

$param$"2":
      G.Dis    Mid.Pos     G.App
 -0.1133705 -0.5295308 0.6429013
```

For weighted cells, the help file for `loglin` says, “The start argument can be used to produce analyses when the cells are assigned different weights, see Clogg and Eliason (1988). The start should be one over the weights.”

Chapter 7 - Agresti

A. Stepwise Model Selection

As discussed above, the function `step` is used for stepwise selection of glm models. Criteria for selection are Mallow's C_p and RSS. The component "anova" gives a summary of the trace path. To incorporate the sampling design in a stepwise selection, one could make use of the scope option, which allows one to give a lower and upper model.

To illustrate stepwise procedures, I follow Agresti's Section 7.2.4, using the data in Table 7.3-5 (p. 219).

data:

```
table.7.3<-expand.grid(EMS=c("yes","no"),PMS=c("yes","no"),Gender=c("women","men"),
   M.Status=c("divorced","married"))
table.7.3<-
   cbind.data.frame(table.7.3,count=c(17,54,36,214,28,60,17,68,4,25,4,322,11,
   42,4,130))
```

Forward stepwise:

Fit the independence model (this is our initial model)

```
fit<-glm(count~(EMS+PMS+Gender+M.Status),data=table.7.3,family=poisson,link="log")
res<-step(fit,direction="forward",scope=list(lower=formula(fit),
   upper=formula(count~(EMS+PMS+Gender+M.Status)^3)), trace=T,fit=T)
res$anova
```

```
  Resid. Df Resid. Dev      AIC
1        11   232.1398 242.1398
2        10   156.8804 168.8804
3         9    92.4739 106.4739
4         8    46.4576  62.4576
5         7    21.0676  39.0676
6         6     8.1535  28.1535
7         5     5.2463  27.2463
8         4     0.7569  24.7569
```

Compare this table to Table 7.4.

Backward stepwise:

Fit initial model - all 2-way interactions:

```
fit<-glm(count~(EMS+PMS+Gender+M.Status)^3,data=table.7.3,family=poisson,link="log")
res<-step(fit,direction="backward",
   scope=list(lower=formula(count~(EMS+PMS+Gender+M.Status)),
   upper=formula(count~(EMS+PMS+Gender+M.Status)^3)),trace=T,fit=T)
res$anova
```

```
  Resid. Df Resid. Dev      AIC
1         1  0.1463554 30.14636
2         2  0.1854976 28.18550
3         3  0.3674045 26.36740
4         4  0.7569237 24.75692
```

Compare output to Table 7.5.

B. Adjusting a model using indicator functions

I use the data in Table 7.7 (p. 226) - Graduate school admissions - for this section.

```
table.7.7<-expand.grid(admit=c("yes","no"),gender=c("male","female"),
   dept=LETTERS[1:6])
table.7.7<-cbind.data.frame(table.7.7,count=c(512,313,89,19,353,207,17,8,120,
   205,202,391,138,279,131,244,53,138,94,299,22,351,24,317))
```

I will fit the model in equation (7.7).

Start with model (AD,DG,GA):

```
fit<-glm(count~.^2, data=table.7.7, family=poisson, link="log")
X<-model.matrix(fit)
```

Make the indicator in equation (7.7)

```
X[table.7.7[,"dept"]=="A",][,"admit:gender"] <-1
X[table.7.7[,"dept"]!="A",][,"admit:gender"] <-0
ind<-X[,"admit:gender"]
```

Now, fit the new model

```
fit<-glm(count~ gender:admit:ind + dept*admit + dept*gender, data=table.7.7,
   family=poisson, link="log")
```

```
Degrees of Freedom: 24 Total; 5 Residual
Residual Deviance: 2.681497
```

To obtain fitted values in an array, use the following command (notice dept. A):

```
array(fitted(fit) ,dim=c(2,2,6), dimnames=list(c("yes","no"), c("male","female"),
   LETTERS[1:6]))
```

```
, , A
    male female
yes  512     89
 no  313     19

, , B
       male    female
yes 354.188 15.811966
 no 205.812  9.188034

, , C
        male   female
yes 113.9978 208.0022
 no 211.0022 384.9978

, , D
        male   female
yes 141.6326 127.3674
 no 275.3674 247.6326

, , E
         male    female
yes  48.07705  98.92295
 no 142.92295 294.07705

, , F
         male    female
yes  24.03081  21.96919
 no 348.96919 319.03081
```

The same analysis can be done using the suggestion in section 7.3.4 (p.228) with structural zeroes. `loglin` (and `loglm`) will handle structural zeroes if you specify initial values of 0 for the appropriate cells and 1's for the remaining cells.

Create an array out of table.7.7
`table.7.17<-design.table(table.7.7)`

Repeat the (AD,DG) fit:
`loglin(table.7.17,margin=list(c(1,3),c(2,3)),param=T,fit=T)`

Now add the structural zeros and fit equation (7.7). Notice we start the subtable with structural zeroes at zero. The subtable values remain at zero through the fitting process.

```
table.7.17[,,1]<-0
loglin(table.7.17,margin=list(c(1,3),c(2,3)),start=c(rep(0,4),rep(1,20)),fit=T)

2 iterations: deviation 0
$lrt:
[1] 2.681459

$pearson:
[1] 2.6904

$df:
[1] 6  # Ripley (1997) warns that using loglin with structural zeroes sometimes gives
   incorrect df, as here

$margin:
$margin[[1]]:
[1] "admit" "dept"

$margin[[2]]:
[1] "gender" "dept"


$fit:

, , A
    male female
yes    0      0
 no    0      0

, , B
       male    female
yes 354.188 15.811966
 no 205.812  9.188034

, , C
        male   female
yes 113.9978 208.0022
 no 211.0022 384.9978

, , D
        male   female
yes 141.6326 127.3674
 no 275.3674 247.6326

, , E
         male    female
yes  48.07705  98.92294
 no 142.92294 294.07706
```

```
, , F
         male    female
yes  24.03081  21.96919
 no 348.96918 319.03082
```

Compare the fit with the above example.

C. Cochran-Mantel-Haenszel Test

To demonstrate this test, I use the data from Table 7.9 (p. 233) and the function `mantelhaen.test`.

```
table.7.9<-array(c(0,0,6,5,3,0,3,6,6,2,0,4,5,6,1,0,2,5,0,0), dim=c(2,2,5),
   dimnames=list(delay=c("none","some"),resp=c("cured","died"),
   pen.level=c(1/8,1/4,1/2,1,4)))

mantelhaen.test(table.7.9)  # this takes in variables from a data frame too

Warning messages:
  Warning in mantelhaen.test(table.7.9): Columns in some subtables x[,,k] contain
    only zeroes.

   Mantel-Haenszel chi-square test with continuity correction

data:  table.7.9
Mantel-Haenszel chi-square = 3.9286, df = 1, p-value = 0.0475
```

Table 7.10 (columns 2 & 3)

For column 2: extract the LRT statistic for model (CD,PD,PC)

```
loglin(table.7.9,margin=list(c(1,2),c(2,3),c(1,3)))$lrt
[1] 7.501059
```

For column 3: model (PD,PC)

```
loglin(table.7.9,margin=list(c(2,3),c(1,3)))$lrt
[1] 14.29378
```

Mantel-Haenszel odds ratios can be obtained with the help of the following function, `OR.MH`, applied to an array:

```
OR.MH<-function(x) {
   n11k<-x[1,1,]
   n22k<-x[2,2,]
   n12k<-x[1,2,]
   n21k<-x[2,1,]
   nk<-n11k+n22k+n12k+n21k
   sum(n11k*n22k/nk)/sum(n12k*n21k/nk)
}
```

Applied to Table 7.9, the function `OR.MH` gives the common log odds ratio

```
log(OR.MH(table.7.9))
[1] 1.94591
```

Standard errors can be computed using the same techniques.

D. Sample size and power calculations

Frank Harrell's Hmisc library has many functions for power and sample size calculations. In particular, `bpower` calculates the formula on page 240. Here it is applied to the example at the end of p. 240.

```
bpower(p1=.63,p2=.57,n=50)  # .63-.57 = .06 absolute difference
      Power
 0.07128458

bpower(p1=.63,p2=.57,n=200)
     Power
 0.1388583
```

Power for a chi-squared test can be computed using the following function:

```
chipower.f<-function(p,pm,n,alpha,df)
{
   nc<-n*my.pearson.x2(observed=p,expected=pm)$X2
   1-pchisq(qchisq(1-alpha,df),df=df,ncp=nc)
}
```

I demonstrate its use with the example at the end of p. 241.

```
fit<- loglin(matrix(c(.315,.185,.285,.215),byrow=T,ncol=2),margin=list(1,2),fit=T)
chipower.f(p=matrix(c(.315,.185,.285,.215),byrow=T,ncol=2), pm=fit$fit,
       n=200,alpha=.05,df=fit$df)

[1] 0.1393477
```

Chapter 8 - Agresti

A. Uniform Association model

I will use the Job satisfaction data (table.2.4) for this section.

```
      income      jobsat count
 1     <6000 very dissat    20
 2 upto15000 very dissat    22
 3 upto25000 very dissat    13
 4    >25000 very dissat     7
 5     <6000      little    24
 6 upto15000      little    38
 7 upto25000      little    28
 8    >25000      little    18
 9     <6000    moderate    80
10 upto15000    moderate   104
11 upto25000    moderate    81
12    >25000    moderate    54
13     <6000    very sat    82
14 upto15000    very sat   125
15 upto25000    very sat   113
16    >25000    very sat    92
```

I add equal interval scores for income and jobsat to the table using the function `codes` to get the scores:

```
attach(table.2.4)
table.2.4.new<-cbind.data.frame(table.2.4,c.income=codes(ordered(income)),
   c.jobsat=codes(ordered(jobsat)) )
```

I fit the Uniform Association model using scores for the variables only in the interaction. Note the use of ':' in the interaction term.

```
fit<-glm(count ~ income + jobsat + c.income:c.jobsat, data=table.2.4.new,
   family=poisson,link="log")

Coefficients:
 (Intercept)     income1    income2    income3   jobsat1  jobsat2   jobsat3
    3.025165 -0.00526504 -0.1905522 -0.2670179 0.1537626 0.325433 0.1581947

 c.income:c.jobsat
         0.1119394

Degrees of Freedom: 16 Total; 8 Residual
Residual Deviance: 2.385921
```

Notice the coefficient on `c.income:c.jobsat`. This is Agresti's beta-hat.

To obtain a test of the association parameter, beta, use the `summary` command:

```
summary(fit)$coef["c.income:c.jobsat",]

     Value Std. Error  t value
 0.1119394 0.03640746 3.074628
```

Obtain the fitted values for the uniform association model using:

```
matrix(fitted(fit),byrow=F,ncol=4,dimnames=list(Income=c(levels(income)),JobSat=c(le
   vels(jobsat))))
```

```
          very dissat   little  moderate  very sat
    <6000   19.347234 29.42999  74.92518  82.29760
upto15000   21.412156 36.42892 103.72863 127.43030
upto25000   13.592345 25.86395  82.36865 113.17505
   >25000    7.648265 16.27714  57.97755  89.09704
```

Now, Agresti fits the Independence model and compares it to the Uniform Association model.

Fit the Independence model:

```
fit2<-glm(count ~ income + jobsat, data=table.2.4.new, family=poisson,link="log")
```

Compare the two models (Uniform Association v. Independence):

```
anova(fit2,fit)
```

```
Analysis of Deviance Table

Response: count

                              Terms Resid. Df Resid. Dev              Test Df
1                     income + jobsat         9   12.03686
2 income + jobsat + c.income:c.jobsat         8    2.38592 +c.income:c.jobsat  1

  Deviance     Pr(Chi)
1
2 9.650942 0.001892545
```

The deviance above is G-squared for the Independence model given uniform association holds.

B. Row Effects model - (same effect for an entire row)

Agresti fits a row effects model to the data in Table 8.4 - p. 273.

```
table.8.4<-data.frame(expand.grid(Affil=c("Democrat","Independent","Republican"),
  Ideology=c("Liberal","Moderate","Conservative")),
   c.Ideo=codes(ordered(rep(c(1,2,3),c(3,3,3)))),
   count=c(143,119,15,156,210,72,100,141,127))
attach(table.8.4)
```

To fit Row Effects model, type:

```
fit<-glm(count~Affil+Ideology+c.Ideo:Affil,family=poisson,link="log",data=table.8.4)
```

```
Degrees of Freedom: 9 Total; 2 Residual
Residual Deviance: 2.814931
```

To obtain fitted values for the Row Effects model, type:

```
matrix(fitted(fit),byrow=F,ncol=3,dimnames=list(Affiliation=c(levels(Affil)),Ideolog
   y=c(levels(Ideology))))
```

```
             Liberal  Moderate Conservative
   Democrat 136.63414 168.73171     93.63414
Independent 123.79454 200.41091    145.79454
 Republican  16.57131  68.85738    128.57131
```

To obtain the row effects themselves, we use the interaction parameter estimates.

```
res<-coef(fit)

 (Intercept)     Affil1    Affil2 Ideology1 Ideology2 AffilDemocratc.Ideo
    6.057586 -0.1847135 -1.046094 0.7121822 0.4455428           -1.213361

 AffilIndependentc.Ideo
             -0.9426178
```

mu1-mu2:

```
temp<-res["AffilDemocratc.Ideo"]-res["AffilIndependentc.Ideo"]
```

mu2:

```
mu2<-(1/3)*(res["AffilIndependentc.Ideo"]-temp)

 AffilIndependentc.Ideo
               -0.2239581
```

Getting the ASE is then also a matter of manipulation.

```
res<-summary(fit)$coef
                           Value Std. Error    t value
           (Intercept)  6.0575857 0.14456061  41.903433
                Affil1 -0.1847135 0.09528375  -1.938563
                Affil2 -1.0460939 0.10073181 -10.384940
             Ideology1  0.7121822 0.07042806  10.112194
             Ideology2  0.4455428 0.05381862   8.278601
   AffilDemocratc.Ideo -1.2133612 0.13041946  -9.303529
AffilIndependentc.Ideo -0.9426178 0.12602316  -7.479719
```

SE of mu2:

```
SE2<-res["AffilIndependentc.Ideo","Std. Error"]
SE1<-res["AffilDemocratc.Ideo","Std. Error"]
res<-summary(fit)$cor[6,7]*SE1*SE2
sqrt( ( (4/9)*(SE2^2)+(1/9)*SE1^2 ) - (4/9)*res )

[1] 0.05885542
```

To compare independence and row effects models, use `anova`:

```
fit2<-glm(count~Affil+Ideology,family=poisson,link="log",data=table.8.4)
anova(fit2,fit)

Analysis of Deviance Table

Response: count

                              Terms Resid. Df Resid. Dev             Test Df
      Deviance
1                   Affil + Ideology         4   105.6622
2    Affil + Ideology + c.Ideo:Affil         2     2.8149 + c.Ideo:Affil  2 102.8472
```

Now, I get predicted logits using `fit$linear.predictors`:

```
res<-matrix(fit$linear.predictors,byrow=F,ncol=3)
mat<-matrix(c(res[,2]-res[,1],res[,3]-res[,2]),byrow=T,ncol=3)
           [,1]       [,2]      [,3]
[1,]  0.2110031  0.4817465 1.4243643
[2,] -0.5889149 -0.3181714 0.6244464
```

You can use the function `matplot` to create Figure 8.2 (p. 274).

C. Modeling three-factor interaction

Agresti uses the data in Table 8.10 p. 280 (effect of smoking) to illustrate modeling a three-factor interaction.

```
table.8.10<-expand.grid(Status=c("Never","Former","Current"),
   Breath=c("Normal","Borderline","Abnormal"), Age=c("< 40","40-50"))
table.8.10<-cbind.data.frame(table.8.10,count=c(577,192,682,27,20,46,7,3,11,164,
   145,245,4,15,47,0,7,27))
attach(table.8.10)
table.8.10<-cbind.data.frame(table.8.10,c.Status=codes(ordered(Status)),
   c.Breath=codes(ordered(Breath)),c.Age=codes(ordered(Age)) )
```

I fit model (8.16) to Table 8.10 using:

```
fit<-glm(count~Breath + Status*Age + c.Status:c.Breath + c.Age:c.Breath +
   c.Age:c.Breath:c.Status, data=table.8.10, family=poisson, link="log")
```

```
Degrees of Freedom: 18 Total; 7 Residual
Residual Deviance: 10.81581
```

To recreate Table 8.11, use the following set of commands:

```
res<-summary(fit,cor=F)$coefficients
```

```
Coefficients:
                            Value Std. Error    t value
            (Intercept)  4.69087634 1.15591459  4.0581514
                Breath1 -0.98551429 0.28466131 -3.4620591
                Breath2 -0.57776113 0.28454270 -2.0304901
                Status1 -0.50384721 0.05397706 -9.3344686
                Status2 -0.03229618 0.04871983 -0.6628959
                    Age -0.58570923 0.08588420 -6.8197550
             Status1Age  0.08054713 0.05378014  1.4977114
             Status2Age -0.20715182 0.04848249 -4.2727143
      c.Status:c.Breath -0.54788559 0.22174479 -2.4707935
         c.Age:c.Breath -1.01492541 0.43958629 -2.3088195
c.Age:c.Breath:c.Status  0.66312565 0.16467956  4.0267635
```

The coefficients from the fitted model (fit as stated in the command) do not correspond exactly to those in Agresti. Instead they are linear combinations of them as follows:

```
beta1<-res["c.Status:c.Breath","Value"]
beta3<-res["c.Age:c.Breath:c.Status","Value"]
beta2<-res["c.Age:c.Breath","Value"]
smoke<-beta1+beta3
Age<-(beta2 + 2*beta3)
smokeXage<-beta3
SEsmoke <- sqrt( (res["c.Status:c.Breath","Std. Error"])^2 +
   res["c.Age:c.Breath:c.Status","Std. Error"]^2 +
   2*summary(fit)$cor[9,11]*(res["c.Status:c.Breath","Std. Error"]*
   res["c.Age:c.Breath:c.Status","Std. Error"]) )
SEAge <- sqrt( (res["c.Age:c.Breath","Std. Error"])^2 +
   4*res["c.Age:c.Breath:c.Status","Std. Error"]^2 +
   2*2*summary(fit)$cor[10,11]* (res["c.Age:c.Breath","Std.
   Error"]*res["c.Age:c.Breath:c.Status","Std. Error"]) )

matrix(c(smoke,Age,smokeXage,SEsmoke,SEAge,res["c.Age:c.Breath:c.Status","Std.
   Error"]),byrow=F,ncol=2, dimnames=list(Parameter=c("Smoking","Age","Smoking x
   age"),c("Estimate","SE")))
```

```
               Estimate         SE
      Smoking 0.1152401 0.08601739
          Age 0.3113259 0.15107196
Smoking x age 0.6631256 0.16467956
```

D. Uniform interaction model

The following command fits a uniform interaction model to the data in Table 8.10:

```
fit2<-glm(count~(Status+Age+Breath)^2 + c.Breath:c.Age:c.Status, data=table.8.10,
   family=poisson, link="log")

Degrees of Freedom: 18 Total; 3 Residual
Residual Deviance: 2.744473

summary(fit2,cor=F)$coefficients["c.Breath:c.Age:c.Status",]

     Value Std. Error  t value
 0.8311125  0.1945378 4.272242
```

E. Mantel Score Test of Conditional Independence

To illustrate Mantel's Score Test, Agresti uses the data in Table 8.13.

```
dose<-(c(0,5,30,75,150,300))
survive<-c(0,1)
nijk<-array(c(0,7,3,1,4,11,5015,10752,2989,694,418,387,5,4,6,1,3,6,5973,11811,
   2620,771,792,820,2,8,3,1,3,7,5669,10828,2798,797,596,624,3,19,4,2,1,10,6158,12645
   ,3566,972,694,608,3,7,3,2,2,6,3695,9053,2415,655,393,289),dim=c(6,2,5))
```

I transpose each stratum (so that they look like Table 8.13)

```
nijk<-array(apply(nijk,3,t),dim=c(2,6,5))
```

Now, get the marginal sums:

```
n.jk<-array(apply(nijk,3,rowSums),c(2,1,5))
ni.k<-array(apply(nijk,3,colSums),c(1,6,5))
n..k<-apply(nijk,3,sum)
```

Get intermediate quantities in order to compute equation (8.17):

```
u.k<-apply(ni.k,3,function(x,u){u%*%t(x)},u=dose)
v.k<-apply(n.jk,3,function(x,v){v%*%x},v=survive)
E.k<-(u.k)*(v.k)/n..k
u2.k<-apply(ni.k,3,function(x,u){(u^2)%*%t(x)},u=dose)
v2.k<-apply(n.jk,3,function(x,v){(v^2)%*%x},v=survive)
var.k<- (1/n..k)*(u2.k-((u.k)^2)/n..k)*(v2.k-(v.k^2)/n..k)
grid<-as.matrix(survive)%*%dose
uv.k<-colSums(apply(nijk,3,function(x,uv){x*uv},uv=grid))
```

Equation (8.17):

```
((sum(uv.k-E.k))^2)/sum(var.k)
[1] 397.0276
```

The answer should be 426.3. So, there is an error somewhere..

F. Correspondence Analysis

The mutliv library contains a function, ca, which performs correspondence analysis. To add the library, type

```
library(multiv)
```

I will use the data in Table 8.14 to follow Agresti's example in Section 8.5.4.

```
table.8.14<-data.frame(expand.grid(MH=c("well","mild","moderate","impaired"),
   SES=c(LETTERS[1:6])),
   count=c(64,94,58,46,57,94,54,40,57,105,65,60,71,141,77,94,36,97,54,78,
   21,71,54,71))
attach(table.8.14)
```

```
          MH SES count
 1     well   A    64
 2     mild   A    94
 3 moderate   A    58
 4 impaired   A    46
 5     well   B    57
 6     mild   B    94
 7 moderate   B    54
 8 impaired   B    40
. . .
```

I now put the data frame above into an array and transpose it:

```
temp<-t(design.table(table.8.14))
```

To do the correspondence analysis, I use the ca function:

```
fit<-ca(temp, nf=3)
```

Now, I extract the squared correlations for the three dimensions

```
fit$evals
[1] 0.0260861516 0.0013649208 0.0002818179
```

To get the row and column projections (i.e., Table 8.15), use the following columns:

column projections

```
fit$cproj
       "Factor1"   "Factor2"    "Factor3"
[1,]  0.26031950  0.01059587  0.022159025
[2,]  0.02943672  0.02487361 -0.018911108
[3,] -0.01373850 -0.06926306 -0.004133451
[4,] -0.23754509  0.01763202  0.015694946
```

row projections

```
fit$rproj
       "Factor1"   "Factor2"    "Factor3"
[1,]  0.18204767 -0.01965235  0.027711924
[2,]  0.18603650 -0.01028790 -0.026940031
[3,]  0.06004478 -0.02157942 -0.010481284
[4,] -0.01218749  0.04121657  0.009749501
[5,] -0.16456713  0.04381708 -0.008189335
[6,] -0.28677478 -0.06237175  0.003613453
```

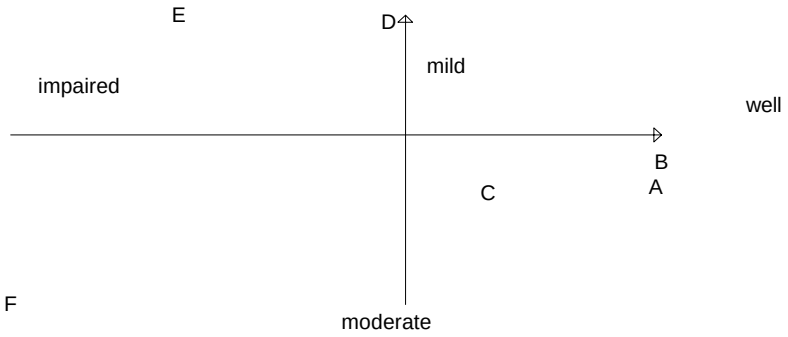
To plot the first and second factors (Figure 8.3), use

```
plot(fit$rproj[,1], fit$rproj[,2],type="n",ylim=c(-.1,.1),xlim=c(-.3,.3),
   xlab="",ylab="",axes=F)
text(fit$rproj[,1], fit$rproj[,2], labels=dimnames(temp)$SES)
text(fit$cproj[,1], fit$cproj[,2], labels=dimnames(temp)$MH)
```

One can place additional axes through x=0 and y=0 using a modified version of the function `plaxes` in library multiv:

```
my.plaxes.f(fit$rproj[,1], fit$rproj[,2],size=.15)
```

```
my.plaxes.f<-
function(a, b, size = 0.1)
{
   arrows(min(a), 0, max(a), 0, size = size)
   arrows(0, min(b), 0, max(b), size = size)
}
```



Note that the MASS library also has a multivariate CA function (as well as a univariate one). It is called `mca`. I had to alter it to fit three factors. With the above dataset, the new function, `my.mca` gives

```
counts<-c(64,94,58,46,57,94,54,40,57,105,65,60,71,141,77,94,36,97,54,78,21,71,54,71)
table.8.14<-apply(table.8.14[,c(1,2)],2,function(x){rep(x,counts)})

fit<-my.mca(as.data.frame(table.8.14),nf=3)
```

MASS has a univariate CA, which estimates a single factor model. Using it on Table 8.14,

```
temp<-t(design.table(table.8.14))  # transposed the default

fit<-corresp(temp)
```

```
$cor:
[1] 0.1615119

$rscore:
         A          B          C          D        E        F
 -1.127146 -1.151843 -0.3717666 0.07545902 1.018916 1.775564

$cscore:
      well       mild   moderate impaired
 -1.611765 -0.1822571 0.08506316 1.470761
```

The rscore and cscore match the scores given on p. 293, for a one-dimensional model.

For canonical correlation model, note that Splus has a cancor function

Chapter 9 - Agresti

A. Generalized logit model for multiple nominal response categories

To fit a generalized logit model for multiple nominal response categories, I will Agresti's use of Table 9.1 (Alligator food choice). To set up the data, type:

```
food.labs<-c("fish","invert","rep","bird","other")
size.labs<-c("<2.3",">2.3")
gender.labs<-c("m","f")
lake.labs<-c("hancock","oklawaha","trafford","george")

table.9.1<-
expand.grid(food=food.labs,size=size.labs,gender=gender.labs,lake=lake.labs)
temp<c(7,1,0,0,5,4,0,0,1,2,16,3,2,2,3,3,0,1,2,3,2,2,0,0,1,13,7,6,0,0,3,9,1,0,2,0,1,0
,1,0,3,7,1,0,1,8,6,6,3,5,2,4,1,1,4,0,1,0,0,0,13,10,0,2,2,9,0,0,1,2,3,9,1,0,1,8,1,0,0
,1)

table.9.1<-apply(table.9.1,2,function(x){rep(x,temp)})
```

```
           food   size gender       lake
  [1,] "fish"   "<2.3" "m"    "hancock"
  [2,] "fish"   "<2.3" "m"    "hancock"
  [3,] "fish"   "<2.3" "m"    "hancock"
  [4,] "fish"   "<2.3" "m"    "hancock"
  [5,] "fish"   "<2.3" "m"    "hancock"
  [6,] "fish"   "<2.3" "m"    "hancock"
  [7,] "fish"   "<2.3" "m"    "hancock"
  [8,] "invert" "<2.3" "m"    "hancock"
  [9,] "other"  "<2.3" "m"    "hancock"
 [10,] "other"  "<2.3" "m"    "hancock"
 [11,] "other"  "<2.3" "m"    "hancock"
 [12,] "other"  "<2.3" "m"    "hancock"
 [13,] "other"  "<2.3" "m"    "hancock"
 [14,] "fish"   ">2.3" "m"    "hancock"
 [15,] "fish"   ">2.3" "m"    "hancock"
 [16,] "fish"   ">2.3" "m"    "hancock"
 [17,] "fish"   ">2.3" "m"    "hancock"
 [18,] "bird"   ">2.3" "m"    "hancock"
 [19,] "other"  ">2.3" "m"    "hancock"
 [20,] "other"  ">2.3" "m"    "hancock"
 [21,] "fish"   "<2.3" "f"    "hancock"
 [22,] "fish"   "<2.3" "f"    "hancock"
 [23,] "fish"   "<2.3" "f"    "hancock"
 [24,] "fish"   "<2.3" "f"    "hancock"
 [25,] "fish"   "<2.3" "f"    "hancock"
 [26,] "fish"   "<2.3" "f"    "hancock"
 [27,] "fish"   "<2.3" "f"    "hancock"
 [28,] "fish"   "<2.3" "f"    "hancock"
...
```

First, I collapse over gender (as on page 310):

```
table91.nogender<-table(category(table.9.1[,1],levels=food.labs),
   category(table.9.1[,2],levels=size.labs),category(table.9.1[,4],levels=lake.labs)
   )
```

(The reason for using the `category` function above is to give the labels in an order that is not alphabetical)

Now, I first fit the model (FS, FL, LS) (see Table.9.3):

```
fit<-loglin(table91.nogender,margin=list(c(1,3),c(1,2),c(2,3)),fit=T)

$lrt:
[1] 17.08009

$pearson:
[1] 15.0311

$df:
[1] 12

$margin:
$margin[[1]]:
[1] 1 3

$margin[[2]]:
[1] 1 2

$margin[[3]]:
[1] 2 3


$fit:

, , hancock
            <2.3      >2.3
  fish 20.872540 9.1182861
invert  3.635672 0.3709036
   rep  1.851167 1.1534176
  bird  2.744344 2.2513630
 other  9.896276 3.1060295

, , oklawaha
             <2.3      >2.3
  fish  5.1804056 12.839643
invert 12.0193529  6.956785
   rep  1.5437055  5.457033
  bird  0.1771461  0.824498
 other  1.0793886  1.922041

, , trafford
             <2.3     >2.3
  fish  4.4332786 8.575500
invert 12.3886242 5.596234
   rep  2.1282732 5.871712
  bird  0.8644169 3.139977
 other  4.1854053 5.816578

, , george
             <2.3       >2.3
  fish 18.5324135 14.4535551
invert 16.9330711  3.0840175
   rep  0.4735392  0.5267465
  bird  1.2163639  1.7814560
 other  3.8446136  2.1542249
```

To fit the model as a generalized logit model, we can use B. Ripley's `multinom` function in the nnet library. This function fits multinomial logit models with nominal response categories.

To use the function `multinom`, we transform the format of the data, collapsing over Gender. It is important that the level "fish" be coded as 1. Thus, I use an ordered factor instead of a factor. This will not affect the analysis.

```
library(nnet)
```

```
table.9.1<-data.frame(food=codes(ordered(table.9.1[,"food"],lev=food.labs)),
   table.9.1[,c(2,4)])
```

Now, set the contrasts option so that coefficient estimates sum to zero:

```
options(contrasts=c("contr.sum","contr.poly"))
```

Fit the model:

```
fit<-multinom(food~lake+size,data=table.9.1)

summary(fit,cor=F)$coefficients
```

```
  (Intercept)      lake1      lake2      lake3       size
2  -0.7197536 -0.1001507 -1.7585930  0.8370343  0.7290775
3  -1.8309445 -1.6592416 -0.4164767  0.7996359 -0.1755950
4  -2.1260480 -0.2824301  0.4127130 -0.9356402 -0.3154096
5  -1.1514310 -0.5870533  0.2391164 -0.5814067  0.1657686
```

Note that the lakes are numbered in a different order than that in Table 9.4.

Conditional logit models can be handled by `multinom` or by the function `coxph`, for Cox proportional hazards models. Also, all of the models discussed in section 9.2 can be handled by `multinom`.

B. Multinomial Logits for Ordinal Responses (Continuation-Ratio logit model)

For this section, I will use the set of libraries Design and Hmisc by F. Harrell. In particular, the library fits forward continuation-ratio models and proportional odds models, and many other models which use multinomial ordinal responses.

Continuation-Ratio logit model

For this model, I use the data in Table 9.7 (p. 320), following Agresti's example. A continuation-ratio model is fit using `cr.setup` and `lrm` from the Design library.

First, it is important to add the libraries in this order:

```
library(Hmisc,T)
library(Design,T)
```

Now, set up the data:

```
y<-ordered(c("non-live","malformed","normal"),levels=c("non-
   live","malformed","normal"))
y<-rep(rep(y,5),c(15,1,281,17,0,225,22,7,283,38,59,202,144,132,9))
x<-rep(c(0,62.5,125,250,500),c(15+1+281,17+0+225,22+7+283,38+59+202,144+132+9))

table.9.7<-table(x,y)
```

```
     non-live malformed normal
   0       15         1    281
62.5       17         0    225
 125       22         7    283
 250       38        59    202
 500      144       132      9
```

`cr.setup` will transform the response variable so that it can be used for a continuation-ratio model. In particular, it will create the new variables, y and cohort. The newly created variables are longer (have

more observations) than the old response variable. Cohort defines the denominator categories for each logit (see equation (9.8) in Agresti). y is the transformed response variable taking on values 0 or 1 depending on whether the observation is a success or not within its cohort.

For example, for the data in table 9.7, there are two cohorts. The first cohort (j=1) is the set of all three categories: non-live, malformation, and normal, where an observation is considered a success if it falls in non-live versus either of the other two categories. The second cohort (j=2) is the set of the last two categories, malformed and normal, where an observation is considered a success if it falls in malformed over normal.

Here is how to fit the model:

First set up the response:

```
u<-cr.setup(y)
y.u<-u$y
x.u<-x[u$subs]  # this ensures that the covariate is the correct length
```

I will do separate fits first before showing how to fit both models (j=1 and j=2) together. After the fit of the j=1 model, I will discuss some of the output from `lrm` and compare it to Agresti's results and to results using `glm`.

To fit the j=1 model:

```
fit<-lrm(y.u[u$cohort=="all"]~x.u[u$cohort=="all"])

Logistic Regression Model

Frequencies of Responses
    0   1
 1199 236

  Obs Max Deriv Model L.R. d.f. P     C   Dxy Gamma Tau-a    R2 Brier
 1435    2e-009     253.33    1 0 0.781 0.561 0.667 0.154 0.274 0.108

               Coef      S.E. Wald Z P
Intercept -3.247934 0.1576602 -20.6  0
      x.u  0.006389 0.0004348  14.7  0
```

The "Model L.R." given above is supposed to be the model likelihood ratio chisquare according to Harrell (1998). However, if you examine what `glm` gives and what Agresti says is the model deviance (5.77), you can see that model L.R. is actually equal to -2LogLH(model with intercept + x). The d.f. above gives the number of d.f. used with the addition of x in the model (i.e., 1). What Agresti gives (p. 321) is the model residual deviance. That is, he gives -2LogLH(model with intercept only)-(-2LogLH(model with intercept + x). His d.f. correspond to the resulting d.f. when going from an intercept model (df=4) to a model with x (df=3). These are the df and LR statistic given directly by `glm` when modeling a linear logit, as shown later.

To fit the j=2 model:

```
fit<-lrm(y.u[u$cohort=="y>=malformed"]~x.u[u$cohort=="y>=malformed"])

Logistic Regression Model

Frequencies of Responses
    0   1
 1000 199
```

```
  Obs Max Deriv Model L.R. d.f. P     C   Dxy Gamma Tau-a    R2 Brier
 1199    5e-006     646.52    1 0 0.948 0.895  0.97 0.248 0.703 0.052

              Coef     S.E. Wald Z P
Intercept -5.70190 0.332249 -17.16 0
      x.u  0.01737 0.001227  14.16 0
```

See Harrell (1998) or the help file for the library for a full discussion of the other statistics produced by `lrm`.

To fit both models together (j=1 and j=2), fit an interaction term, as in the following.

```
fit<-lrm(y~cohort*x)

Logistic Regression Model

Frequencies of Responses
    0   1
 2199 435

  Obs Max Deriv Model L.R. d.f. P     C   Dxy Gamma Tau-a    R2 Brier
 2634    4e-006     899.86    3 0 0.884 0.768 0.819 0.212 0.489 0.083

                             Coef      S.E. Wald Z P
             Intercept -3.247934 0.1576602 -20.60 0
    cohort=y>=malformed -2.453968 0.3677581  -6.67 0
                      x  0.006389 0.0004348  14.70 0
cohort=y>=malformed * x  0.010986 0.0013020   8.44 0
```

When y>=malformed (j=2), the linear logit is -5.70 + .017x.

When y=all (j=1), the linear logit is -3.247 + .0064x.

Notice that the values for model L.R.in the individual model sum to the model L.R. for the interaction model above. However, the d.f. do not add.

Odds Ratios

To get selected odds ratios for the j=2 model, first issue the `datadist` command and reissue the `lrm` call, as follows:

```
x.u<-x.u[u$cohort=="y>=malformed"]
dd<-datadist(x.u)
options(datadist='dd')
fit<-lrm(y.u[u$cohort=="y>=malformed"]~x.u)
```

Using `summary(fit)` will give odds ratios comparing the default levels of `x.u` (the lowest and highest nonzero values)

```
summary(fit)
            Effects              Response : y.u[u$cohort == "y>=malformed"]

      Factor  Low High Diff. Effect S.E. Lower 0.95 Upper 0.95
 x.u         62.5 250  187.5  3.26  0.23  2.81       3.71
  Odds Ratio 62.5 250  187.5 25.99    NA 16.56      40.80
```

Thus, given that a fetus was alive, the estimated odds of it being malformed versus normal is 26 times higher when a mouse is exposed to 500 mg/kg per day of the toxic substance than when it is exposed to 62.5 mg/kg per day. The NA for SE is apparently not a mistake. Also the value 3.26 = (.0174)*(250-62.5) is the log odds.

To get an odds ratio comparing specific levels of x, for example comparing levels x=125 and x=250:

```
summary(fit,x=c(125,250))
```

```
             Effects              Response : y.u[u$cohort == "y>=malformed"]

      Factor Low High Diff. Effect S.E. Lower 0.95 Upper 0.95
 x.u         125 250  125   2.17   0.15 1.87        2.47
  Odds Ratio 125 250  125   8.77     NA 6.50       11.85
```

Or, levels x=250 and x=500

```
summary(fit,x=c(250,500))
```

```
             Effects              Response : y.u[u$cohort == "y>=malformed"]

      Factor Low High Diff. Effect S.E. Lower 0.95 Upper 0.95
 x.u         250 500  250    4.34  0.31  3.74        4.95
  Odds Ratio 250 500  250   76.99    NA 42.20      140.48
```

Using `glm`

We can instead estimate the continuation ratio model using `glm`, as we did the linear logit model in chapter 4.

First detach Design and Hmisc, or else **`glm`** will not work right:

```
detach("C:\\Program Files\\splus45\\library\\Design\\_Data")
detach("C:\\Program Files\\splus45\\library\\Hmisc\\_Data")
```

Now set x (the covariate) and the weights for the first linear logit model

```
x<-c(0,62.5,125,250,500)
n1<-rowSums(table.9.7) # use the whole table
```

For model j=1, take the first column of table.9.7 as the “success” , as follows:

```
fit<-glm(table.9.7[,1]/n1~x, family=binomial,link=logit,weights=n1)

Coefficients:
 (Intercept)           x
   -3.247934 0.006389069

Degrees of Freedom: 5 Total; 3 Residual
Residual Deviance: 5.777478

fit$null.deviance
[1] 259.1073

fit$null.deviance-fit$deviance   # this is what lrm gave us as model L.R.
[1] 253.3298

summary(fit)

    Null Deviance: 259.1073 on 4 degrees of freedom

Residual Deviance: 5.777478 on 3 degrees of freedom
```

The difference of the above sets of values gives Null Deviance-Residual Deviance = 253.3298 and df=1. These are the Model L.R. and Model df reported by `lrm`.

For the j=2 model, take the second and third columns of table.9.7, and use the second column as the success:

```
n2<-rowSums(table.9.7[,c(2,3)])

glm(table.9.7[,2]/n2~x, family=binomial,link=logit,weights=n2)

Coefficients:
 (Intercept)          x
   -5.701891 0.01737464

Degrees of Freedom: 5 Total; 3 Residual
Residual Deviance: 6.060908
```

Even though `glm` can also fit proportional odds models, there are advantages to using `lrm` for these types of models because of the built-in features, like the odds ratios above. See Harrell(1998) or the web site http://hesweb1.med.virginia.edu/biostat (under Statistical Computing Tools) for more information.

As mentioned in Harrell (1998) as well as Agresti (chapter 9 notes), the continuation-ratio model is a discrete version of the Cox proportional hazards model. Thus, one could probably fit these models using either `coxph`, which comes with Splus, or `cph`, which is in the Design library.

C. Cumulative Logit Models

For these models (proportional odds, uniform association for cumulative odds, row effects model for nominal explanatory variables), we will use the Design library again. To load the library, issue the following commands in the order specified.

```
library(Hmisc, T)
library(Design, T)
```

1. Proportional Odds Model

For this model I use the data in Table 9.8. To set it up, type

```
mental.imp<-rep(c("well","mild","moderate","impaired"),c(12,12,7,9))
mental.imp<-ordered(mental.imp,levels=rev(c("well","mild","moderate","impaired")))
SES<--c(1,1,1,1,0,1,0,1,1,1,0,0,1,0,1,0,1,1,0,1,1,0,1,1,0,1,0,0,1,0,0,1,1,1,0,0,
   0,1,0,0)
LE<--c(1,9,4,3,2,0,1,3,3,7,1,2,5,6,3,1,8,2,5,5,9,3,3,1,0,4,3,9,6,4,3,8,2,7,5,4,4,8,
   8,9)

table.9.8<-data.frame(mental.imp=mental.imp,SES=SES,LE=LE)
```

(The minus signs in SES and life events are required to estimate the parameters for the main effects model on page 325).

`lrm` fits proportional odds models automatically if the response is ordinal:

```
lrm(mental.imp~SES+LE,data=table.9.8)

Frequencies of Responses
 impaired moderate mild well
        9        7   12   12
```

```
 Obs Max Deriv Model L.R. d.f.      P     C   Dxy Gamma Tau-a    R2 Brier
  40    8e-010       9.94    2 0.0069 0.705 0.409 0.425  0.31 0.236 0.146

               Coef   S.E. Wald Z      P
y>=moderate  2.2094 0.7210  3.06  0.0022
    y>=mild  1.2128 0.6607  1.84  0.0664
    y>=well -0.2819 0.6423 -0.44  0.6607
        SES -1.1112 0.6109 -1.82  0.0689
         LE  0.3189 0.1210  2.64  0.0084
```

To add the interaction, we want to prevent the minuses from canceling out in the interaction between SES and LE. So, we can use the `I` function, which interprets its argument literally:

```
lrm(mental.imp ~ SES + LE + I( - SES * LE), data = table.9.8)
```

```
Logistic Regression Model

Frequencies of Responses
 impaired moderate mild well
        9        7   12   12

 Obs Max Deriv Model L.R. d.f.      P     C   Dxy Gamma Tau-a    R2 Brier
  40    1e-009      10.54    3 0.0145 0.706 0.413 0.428 0.313 0.248 0.146

                Coef   S.E. Wald Z      P
y>=moderate  2.60659 0.9035  2.88  0.0039
    y>=mild  1.59248 0.8396  1.90  0.0579
    y>=well  0.09806 0.8195  0.12  0.9047
        SES -0.37089 1.1361 -0.33  0.7441
         LE  0.42045 0.1864  2.26  0.0241
            -0.18131 0.2383 -0.76  0.4468
```

The last coefficient is the interaction term.

To get odds ratios for this last model (the interaction model), we have to explicitly give the interaction term a name in `datadist`, and we should also note that the levels of LE and SES are negated:

```
IA<-I(-SES*LE)  # name the interaction
dd<-datadist(SES=-SES,LE=-LE,IA=IA) # register variables in datadist
options(datadist='dd')
fit<-lrm(mental.imp~SES+LE+IA)   # fit the model

summary(fit)
```

```
Effects              Response : mental.imp

      Factor   Low High Diff. Effect S.E. Lower 0.95 Upper 0.95
 SES          0.00 1.00 1.00  -0.37  1.14 -2.60       1.86
  Odds Ratio  0.00 1.00 1.00   0.69    NA  0.07       6.40
 LE           2.00 6.25 4.25   1.79  0.79  0.23       3.34
  Odds Ratio  2.00 6.25 4.25   5.97    NA  1.26      28.22
 IA          -4.25 0.00 4.25  -0.77  1.01 -2.76       1.21
  Odds Ratio -4.25 0.00 4.25   0.46    NA  0.06       3.37
```

See the function `popower` in library Hmisc for computing power of proportional odds models.

2. Cumulative logit uniform association model

I use the data in Table 9.9 in Agresti and follow Section 9.4.5. To set up the data, type:

```
operation<-factor(c("A","B","C","D"))
operation<-rep(rep(operation,12),c(23,23,20,24,7,10,13,10,2,5,5,6,18,18,13,9,6,
   6,13,15,1,2,2,2,8,12, 11,7,6,4,6,7,3,4,2,4,12,15,14,13,9,3,8,6,1,2,3,4))
```

```
dump<-ordered(c("N","S","M"),levels=c("N","S","M"))
dump<-rep(rep(dump,4),c(23+23+20+24,7+10+13+10,2+5+5+6,18+18+13+9,6+6+13+15,
   1+2+2+2,8+12+11+7,6+4+6+7,3+4+2+4,12+15+14+13,9+3+8+6,1+2+3+4))
hospital<-factor(c("H1","H2","H3","H4"))
hospital<-
   rep(hospital,c(23+23+20+24+7+10+13+10+2+5+5+6,18+18+13+9+6+6+13+15+1+2+2+2,
   8+12+11+7+6+4+6+7+3+4+2+4,12+15+14+13+9+3+8+6+1+2+3+4))
table.8.6<-data.frame(operation=operation,dump=dump,hospital=hospital)

table(operation,dump,hospital)

, , H1
   N  S M
A 23  7 2
B 23 10 5
C 20 13 5
D 24 10 6

, , H2
   N  S M
A 18  6 1
B 18  6 2
C 13 13 2
D  9 15 2

, , H3
   N S M
A  8 6 3
B 12 4 4
C 11 6 2
D  7 7 4

, , H4
   N S M
A 12 9 1
B 15 3 2
C 14 8 3
D 13 6 4
```

To fit model (9.16) (uniform association model), I use scores (codes) for operation and exclude hospital:

```
lrm(dump~I(-codes(operation)),data=table.8.6)

Logistic Regression Model

Frequencies of Responses
   N   S  M
 240 129 48

 Obs Max Deriv Model L.R. d.f.      P     C   Dxy Gamma Tau-a    R2 Brier
 417    4e-008       6.61    1 0.0102 0.564 0.128  0.17 0.072 0.019 0.241

             Coef    S.E. Wald Z      P
     y>=S -0.8820 0.24966 -3.53  0.0004
     y>=M -2.6362 0.28389 -9.29  0.0000
operation -0.2247 0.08798  2.55  0.0106
```

The model L.R. and df give the reduction in G^2 and in df from the fit model and from one assuming independence (ie., it is a test that the coefficient on operation is zero).

To obtain the predicted logits, probabilities, and expected frequencies, we can use the `linear.predictor` attribute (or, `predict.lrm` function) to get linear predictors, and take the anti-logit to

get cumulative probabilities. However, you must note that `lrm` fits the probabilities, $\Pr(y > j)$ (or, $1 - \Pr(y \leq j)$), not $\Pr(y \leq j)$. Thus, to reproduce Table 9.9, I type

```
probs<-unique(exp(fit$linear.predictors)/(1+exp(fit$linear.predictors)))
```

```
[1] 0.3413441 0.3935044 0.4482114 0.5042011
```

```
cumprobs<-1-probs
```

```
[1] 0.6586559 0.6064956 0.5517886 0.4957989
```

```
rowSums(table(operation,dump))*(cumprobs)
```

```
       A        B        C        D
63.23096 63.07554 60.69675 53.05048
```

to get the first column of Table 9.9. The rest follow similarly.

3. Cumulative logit row effects model

Fitting the row effects is just a matter of adding hospital to the formula.

```
fit<-lrm(dump~I(-codes(operation))+hospital,data=table.8.6)
```

```
Logistic Regression Model

Frequencies of Responses
   N   S  M
 240 129 48

 Obs Max Deriv Model L.R. d.f.      P     C   Dxy Gamma Tau-a    R2 Brier
 417    2e-007       9.16    4 0.0572 0.575 0.149 0.159 0.084 0.026 0.239

                Coef    S.E. Wald Z      P
       y>=S -0.99551 0.28612 -3.48  0.0005
       y>=M -2.75744 0.31808 -8.67  0.0000
  operation -0.22666 0.08814  2.57  0.0101
hospital=H2  0.10383 0.25037  0.41  0.6783
hospital=H3  0.42884 0.28066  1.53  0.1265
hospital=H4  0.02693 0.26901  0.10  0.9202
```

Note that the model L.R. = 31.64-22.48 = 9.16, and d.f. = 30-26= 4

For an interaction model, just use * in the formula:

```
lrm(dump~I(-codes(operation))*hospital,data=table.8.6)
```

D. Cumulative Link Models

Proportional Hazards Models

Although we don't actually have the data in Table 9.10, we can substitute the percentages multiplied by 10 as estimates of the numbers in each category, for illustration purposes. Then, we can use lifelength as the ordered response, or time of death, in a Cox proportional hazards model.

To set up the data, type:

```
lifelength<-ordered(c("0=20","20-40","40-50","50-65","over 65"))
```

```
lifelength<-rep( rep(lifelength,4),c(24,34,38,175,729,36,75,83,250,556,16,14,22,99,
   849,27,29,44,163,737))
race<-c("white","black")
race<-rep(rep(race,2),c(24+34+38+175+729,36+75+83+250+556,16+14+22+99+849,
   27+29+44+163+737))
sex<-rep(c("male","female"),c(24+34+38+175+729+36+75+83+250+556,
   16+14+22+99+849+27+29+44+163+737))
```

To fit a `coxph` model, we need to set up a `Surv` object. The arguments to `Surv` are the time of death for each individual and whether each time was censored (the status). We will assume that all deaths indicated in Table 9.10 were deaths and censor the over-65 category. Thus, our status variable looks like:

```
status<-ifelse(codes(lifelength)<5,1,0)
```

To fit a Cox PH model, type:

```
fit<-coxph(Surv( codes(lifelength), status )~race+sex)
```

```
Call:
coxph(formula = Surv(codes(lifelength), status) ~ race + sex)


       coef exp(coef) se(coef)     z p
race -0.624     0.536   0.0616 -10.1 0
 sex  0.656     1.926   0.0618  10.6 0
```

The coefficients are off somewhat because we didn't use the real data. But, they are roughly the same. (It is important to note that Agresti's *differences* of coefficients are equivalent to `coxph`'s *actual* coefficients). Thus,

$$\Pr(\text{survive until lifelength category } j \mid \text{male}) =$$

$$\Pr(\text{survive until lifelength category } j \mid \text{female})^{\exp(.656)}$$

as in Agresti, p. 333.

E. Regression Model for Ordered Responses

Agresti uses the Olive Preference data in Table 9.11 (p. 335) to illustrate regression on ordered responses, where the conditional mean is linearly related to the explanatory variables. I first tried both of the functions `glm` and `aov`, but can't reproduce the estimates in Agresti exactly. The fitting algorithm used for `glm`, however, is iteratively reweighted least squares, and *not* weighted least squares, as was used by Agresti. I also, then, compute the weighted least squares estimates of the regression coefficients using the development on p. 458-462 of Agresti. These estimates, of course, match those of Agresti.

The first analysis is via the functions `glm` and `aov`.

First, set up the data:

```
location<-factor(c("MW","NE","SW"))
pref<-c(1,2.5,4.5,6,7,8.5)
Urban<-factor(c("Rural","Urban"))
table.9.11<-data.frame(expand.grid(pref=pref,location=location,Urban=Urban),
   counts=c(20,15,12,17,16,28,18,17,18,18,6,25,12,9,23,21,19,30,30,22,21,17,8,12,23,
   18,20,18,10,15,11,9,26,19,17,24))
```

I set the contrasts option so that contrasts on the levels of a factor sum to zero.

```
options(contrasts=c("contr.sum","contr.poly"))
```

Using `glm` we get:

```
fit<-glm(pref~location+Urban,family=poisson(identity),weights=table.9.11$counts,
   data=table.9.11)
fit$coefficients
```

```
(Intercept)  location1  location2    Urban
     4.97535 -0.3923726 -0.2691034 0.330591
```

```
predict(fit)[seq(along=predict(fit))%%6==1]
        1         7        13        19        25        31
 4.913568 5.036838 5.967417 4.252387 4.375656 5.306235
```

Using `aov` we get:

```
fit<-aov(pref ~ Urban + location, data = table.9.11, weights = counts)
fit$coefficients
```

```
 (Intercept)       Urban     location1       location2
 4.975370085  0.3146362183  -0.3715660835  -0.2757134231
```

```
predict(fit)[seq(along=predict(fit))%%6==1]
         1           7          13          19          25          31
 4.91844022 5.014292881 5.93728581 4.289167784 4.385020444 5.308013374
```

The two give similar results. Now, using the development on p. 458 ff., first I set up the data and calculate the sample proportions responding in each preference category by population (i.e., location x urbanization combination).

```
table.9.11<-matrix(c(20,18,12,30,23,11,15,17,9,22,18,9,12,18,23,21,20,26,17,18,21,
   17,18,19,16,6,19,8,10,17,28,25,30,12,15,24), nc=6, nr=6, byrow=F)

n<-rowSums(table.9.11)    # sample size for each population
p<-sweep(table.9.11,1,n,FUN="/")
p1<-p[1,]                 # sample proportions for each of the six categories
p2<-p[2,]
p3<-p[3,]
p4<-p[4,]
p5<-p[5,]
p6<-p[6,]

J<-6   # Six response categories
I<-6   # Six populations
```

Now, I compute the block diagonal variance-covariance matrix, $\mathbf{V}$, using $\mathbf{V}_1$, $\mathbf{V}_2$, $\mathbf{V}_3$, $\mathbf{V}_4$, $\mathbf{V}_5$, $\mathbf{V}_6$ (the population specific covariance matrices). In the for-loop below, the object `Vnames` stores the names of the objects $\mathbf{V}_1$ through $\mathbf{V}_6$, and the object `pnames` stores the names of the vectors of sample proportions for each group. First I calculate each $\mathbf{V}_i$, then I calculate $\mathbf{V}$.

```
Vnames<-numeric(length=I)
pnames<-Vnames
```

```
for(i in 1:I)
{
   Vnames[i] <- paste("V", i, collapse = " ", sep = "")
   pnames[i]<-paste("p",i,collapse=" ",sep="")
   V<-matrix(0,nc=J,nr=J)
   p<-eval(parse(text = pnames[i])) # Use the name 'p' for the current proportion
                                       vector
   diag(V)<-p*(1-p)
   p<-as.matrix(p)
   junk<-matrix(-kronecker(p,p),nc=J,nr=J,byrow=T)
   V[lower.tri(diag(J))]<-junk[lower.tri(junk)]
   V<-t(V)
   V[lower.tri(V)]<-junk[lower.tri(junk,diag=F)]
   assign(Vnames[i], matrix(V/n[i], ncol = J, byrow = T)) # assign V to
                                                            Vnames[i]
}
```

I construct **V** using $\mathbf{V}_1$ through $\mathbf{V}_6$

```
zero<-matrix(0,J,J)
V<-rbind(
cbind(V1,zero,zero,zero,zero,zero),
cbind(zero,V2,zero,zero,zero,zero),
cbind(zero,zero,V3,zero,zero,zero),
cbind(zero,zero,zero,V4,zero,zero),
cbind(zero,zero,zero,zero,V5,zero),
cbind(zero,zero,zero,zero,zero,V6),
)
```

Now, the model for this example has form $F(\boldsymbol{\pi}) = \mathbf{X}\boldsymbol{\beta}$, where $\boldsymbol{\pi}$ is a 6 x 6 matrix of response probability distributions for each population, $\boldsymbol{\beta} = (a,\ b_1^U,\ b_1^L,\ b_2^L)^{\mathrm{T}}$, and $F(\boldsymbol{\pi})$ is a 6 x 1 vector of the 6 response functions

$$F_1(\boldsymbol{\pi}) = a + b_1^U + b_1^L = \boldsymbol{n}^{\mathrm{T}}(p_{1(1,1)},\ldots,p_{6(1,1)}) = \sum_{j=1}^{6} n_j \frac{\exp(a+b_1^U+b_1^L)}{1+\exp(a+b_1^U+b_1^L)}$$

$$\vdots$$

$$F_6(\boldsymbol{\pi}) = a - b_1^U - b_1^L - b_2^L = \boldsymbol{n}^{\mathrm{T}}(p_{1(2,3)},\ldots,p_{6(2,3)}) = \sum_{j=1}^{6} n_j \frac{\exp(a-b_1^U-b_1^L-b_2^L)}{1+\exp(a-b_1^U-b_1^L-b_2^L)}$$

representing the 2 x 3 combinations of Location and Urbanization (see equation 9.18 in Agresti). The 6 x 1 vector $\boldsymbol{n}$ contains the scores for the preference categories.

Now, note that the matrix $\mathbf{Q} = \left[\frac{\partial F_k(\boldsymbol{\pi})}{\partial p_{j|i}}\right]$ on page 459 contains the 1 x 36 row vectors

$$\frac{\partial F_k(\boldsymbol{\pi})}{\partial p_{j|i}} = (0_1,\ldots,0_{(k-1)6},n_1,\ldots,n_6,0,\ldots,0_{36}),$$

and the design matrix is

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & -1 & -1 \end{pmatrix}$$

All of the above is input into Splus via the following commands. For the F( ) functions, we use the sample proportions instead of the probabilities, $\boldsymbol{\pi}$.

```
pref<-c(1,2.5,4.5,6,7,8.5)# the vector nu

Q<-rbind(
   c(pref,rep(0,30)),
   c(rep(0,6),pref,rep(0,24)),
   c(rep(0,12),pref,rep(0,18)),
   c(rep(0,18),pref,rep(0,12)), # derivatives of F
   c(rep(0,24),pref,rep(0,6)),
   c(rep(0,30),pref)
)

VF<-Q%*%V%*%t(Q)    # transformed covariance matrix (p. 460 in Agresti)

# Design matrix:
X<-cbind(rep(1,J),rep(c(1,-1),c(3,3)),rep(c(1,0,-1),2),rep(c(0,1,-1),2))

# Functions
Fp<-c(pref%*%p1,pref%*%p2,pref%*%p3,pref%*%p4,pref%*%p5,pref%*%p6)
```

Now, I estimate beta using the formula in section 13.3.2, the weighted least squares estimator.

```
InvVF<-solve(VF)
Covb<-solve(t(X)%*%InvVF%*%X)
b<-as.numeric(Covb%*%t(X)%*%InvVF%*%Fp)
[1]  4.9680696  0.2965415 -0.3933309 -0.2661780
```

The asymptotic standard errors:

```
sqrt(diag(Covb))
[1] 0.1011683 0.1005547 0.1448348 0.1467234
```

I compute a Wald statistic for entire model using the formula on p. 460 in Agresti.

```
as.numeric(t(Fp-X%*%b)%*%InvVF%*%(Fp-X%*%b))
[1] 4.807433
```

To test H_0: $b_1^L = b_2^L = b_3^L$, we can use the coefficient matrix

```
C<-rbind(c(0,0,1,-1),c(0,0,2,1))
```

and the Wald statistic

```
t(C%*%b)%*%solve(C%*%solve(t(X)%*%solve(VF)%*%X)%*%t(C))%*%C%*%b
23.40537

rm(C)
```

Obtaining predicted mean response values is straightforward using the regression coefficients.

```
X %*% b
             [,1]
[1,] 4.871280128
[2,] 4.998433076
[3,] 5.924119931
[4,] 4.278197184
[5,] 4.405350131
[6,] 5.331036986
```

F. (Supplement) Nonlinear Ordinal Models (Using libraries nolr and lrm)

The following supplement demonstrates the function `nolr`, available in Mathieson's library of the same name. I use the synthetic data provided by Mathieson in the library and compare his analysis in the postscript file (included with the library) with something similar using `lrm`.

For an ordinal model, we might not want to assume that the log odds of $\Pr(y \geq j \,|\, \mathbf{X})$ is a linear function of the vector of covariates, **X**, as is done with the proportional odds models in section 9.4.1 in Agresti. If we do not have a specific form of function in mind, we can fit the model as a feed-forward layered neural network and estimate it using MLE. This can be done using library `nolr`.

A feed-forward neural network is (from Bishop, 1995) a general framework for representing nonlinear functional mappings between a set of input variables and a set of output variables. This is achieved by representing a nonlinear function of a multivariate vector in terms of compositions of nonlinear functions of the scalar elements in the vector. The nonlinear functions are called *activation functions*.

A familiar example of a *single-layered* neural network is logistic discrimination. Let $\mathbf{x}$ be a *d*-dimensional observation assumed to come from one of two Gaussian populations with common covariance. If $a = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$ represents its linear discriminant, and $g(a) \equiv \dfrac{1}{(1+\exp(-a))}$, is a logistic sigmoid activation function, then the output unit $y = g(a)$ is the posterior probability that $\mathbf{x}$ belongs in class 1. Thus, the output units are nonlinear functions of weighted linear combinations of the input units. In the expression for a, w_0 is called the bias term

A neural network with an additional layer of *hidden units* intermediate between the input units and output units is called a multilayered network. The units in a multi-layered feed-forward network have one-way connections to other units, and each unit is only connected to units in higher layers of the network, where the layers are ordered from input to hidden to output. Suppose we have a network with *d* input units, *c* output units and *M* hidden units, where the number of input units represents the dimension of each input variable and the number of output units represents the dimension of each output variable. Following the discussion and notation in Bishop, the output for the *j*th hidden unit is obtained by first forming a weighted linear combination of the *d* input values, and adding a bias to give

$$a_j = \sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

where $w_{ji}^{(1)}$ represents a weight in layer 1 going from input unit *i* to hidden unit *j*. Hidden unit *j* is then *activated* by evaluating a (nonlinear) activation function, $g(\cdot)$, at a_j to get $z_j = g(a_j)$. Then, the output for the *k*th output unit is obtained by first forming a weighted linear combination of the outputs of the *M* hidden units, $a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$, and activating the output unit by evaluating a nonlinear function, $\tilde{g}(\cdot)$, at a_k to get $y_k = \tilde{g}(a_k)$. The entire process of obtaining the output, y_k, from the layered network can be summarized in one expression

$$y_k = \tilde{g}\left(\sum_{j=1}^{M} w_{kj}^{(2)} g\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

Note that the bias terms can be absorbed into the weighted linear combinations by introducing variables, x_0 and z_0, which take the value 1.

Usual choices for the activation functions are linear, logistic, or threshold (step) functions. Also, as might be suspect, the multilayered network can have many intermediate layers, not just one, as considered here.

Using nolr

An example for using `nolr` comes from the author, M. Mathieson in the file example.ps, a postscript file which comes with the library. Mathieson generates a dataset containing patterns from four ordered classes. The object `x.train` (part of the library) is an 80 by 2 matrix of patterns. Each row is a training pattern, and there are 20 from each class. The object `y.train` (also part of the library) is a vector of class labels (i.e., numbers from 1 to 4). Thus, there are 2 input units and 1 output unit. In example.ps, a NOLR model is fit by maximum likelihood using a single-layered network with 3 hidden units. The likelihood is derived in the file example.ps.

```
library(nolr)
net <- nolr(x.train,y,size=3,decay=.1) # fit network
summary(net()

a 2-3-1 network for nolr with 14 weights for 4 classes
options were - skip-layer connections
linear output units
Weight decay = 0.1
Parameters estimated by maximum likelihood

Weights (decays):
  0->3  1->3  2->3  0->4  1->4  2->4
  0.01  0.01 -0.01 -0.20  1.03  2.12
  0->5  1->5  2->5  3->6  4->6  5->6
  2.45  2.06  0.66 -0.01  2.20  3.26
  1->6  2->6
 -1.52  1.36

Cut-points:
[1] -1.271153  2.798747  7.696411

-log(L) =  39.3935594757335
```

The information from `summary(net)` tells us the number of input-hidden-output units we used and the number of weights estimated. The weights are listed under Weights. We are also told that the activation function for the output unit was linear (in nolr, one cannot change this. In `nnet`, one can, to a sigmoid activation function).

The weights are rather cryptic. 0 represents the input layer bias, and 1 and 2 represent the two input units. 3, 4, and 5 represent the hidden units, and 6 is the output unit. Thus the optimal weights and associated paths are shown. The `skip-layer` option allows the network to "skip" the hidden layer and go right from input to output, if necessary. The two weights, 1->6 and 2->6 indicate the weights associated with those paths. The cut-points are defined in the likelihood in the file example.ps.

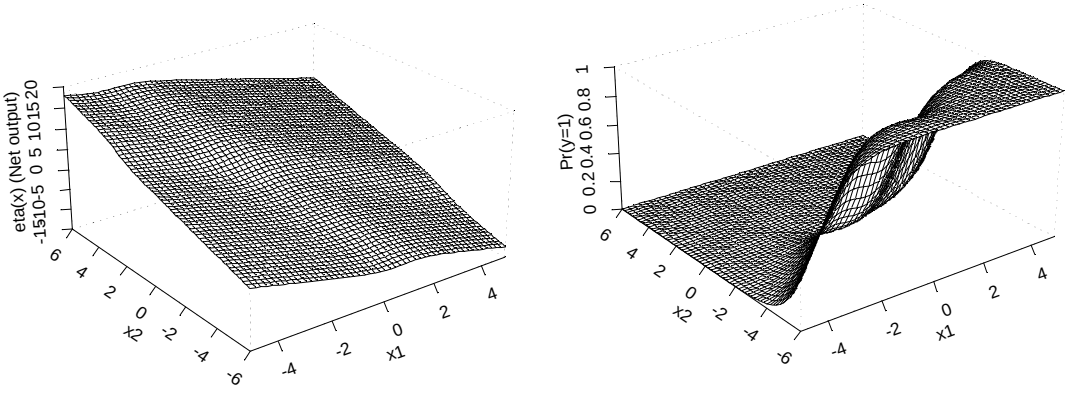
We can plot the 3-dimensional surface indicating the effect of **x** on the output from the network or on the probabilities that **x** comes from a particular class. Below, I show both fitted surfaces. The second plot shows the probability that **x** comes from class 1 (i.e., its response is 1).

The following code produces the plots (see also example.ps)

```
par(mfrow=c(1,2),pty="s")
grid <- list(x=seq(-5,5,0.2),y=seq(-6,6,0.2))
pred <- matrix(predict(net,expand.grid(grid),type="predictor"),
            length(grid$x))
persp(grid$x, grid$y, pred, xlab="x1", ylab="x2",
       zlab="eta(x) (Net output)")

pred <- matrix(predict(net,expand.grid(grid),type="probs")[,1],
            length(grid$x))

persp(grid$x, grid$y, pred, xlab="x1", ylab="x2",
       zlab="Pr(y=1)")
```



Using lrm

To incorporate continuous nonlinear predictors in a ordinal logistic model estimated using `lrm`, one can transform the predictors using a *restricted cubic spline* with *k* knots (a natural cubic spline). For example, using the synthetic data above, we fit a proportional odds model with continuous nonlinear effects of x1 and x2.

In order to take advantage of the `predict.lrm` function, we must do the following two commands (this is part of the use of the Design library only):

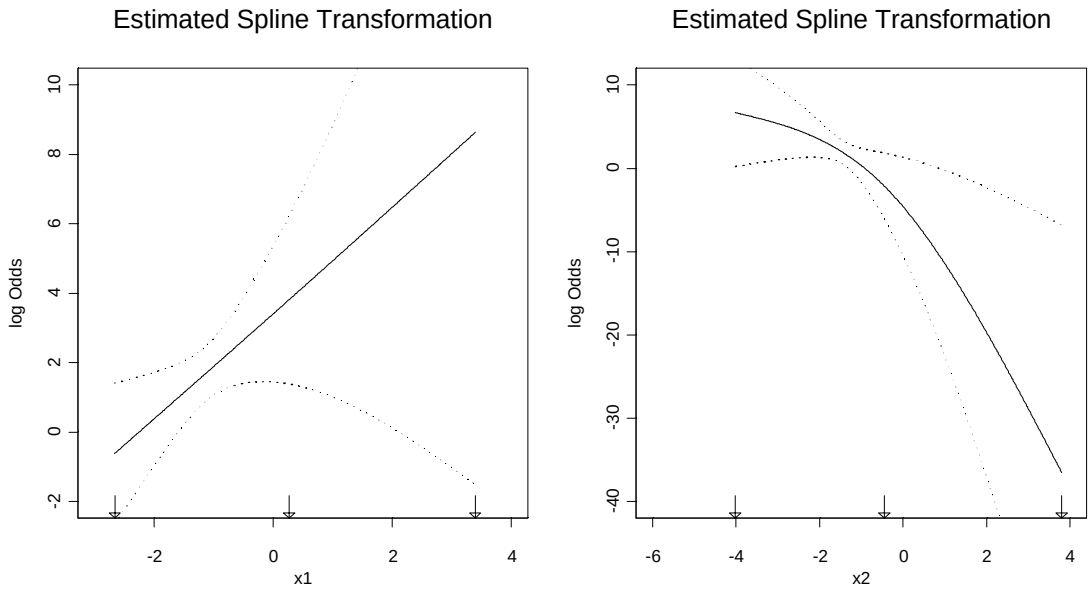
```
library("Hmisc",T)  # if not already in the 2nd search position
library("Design",T)
dd<-datadist(x.train$x1,x.train$x2)
options(datadist='dd')
```

Now fit an additive model with each x element transformed using a restricted cubic spline with 3 knots:

```
fit<-lrm(y.train~rcs(x.train$x1,3) + rcs(x.train$x2,3))
```

To view the marginal effects of the nonlinear transformations on the log odds that y=1, we can plot them.

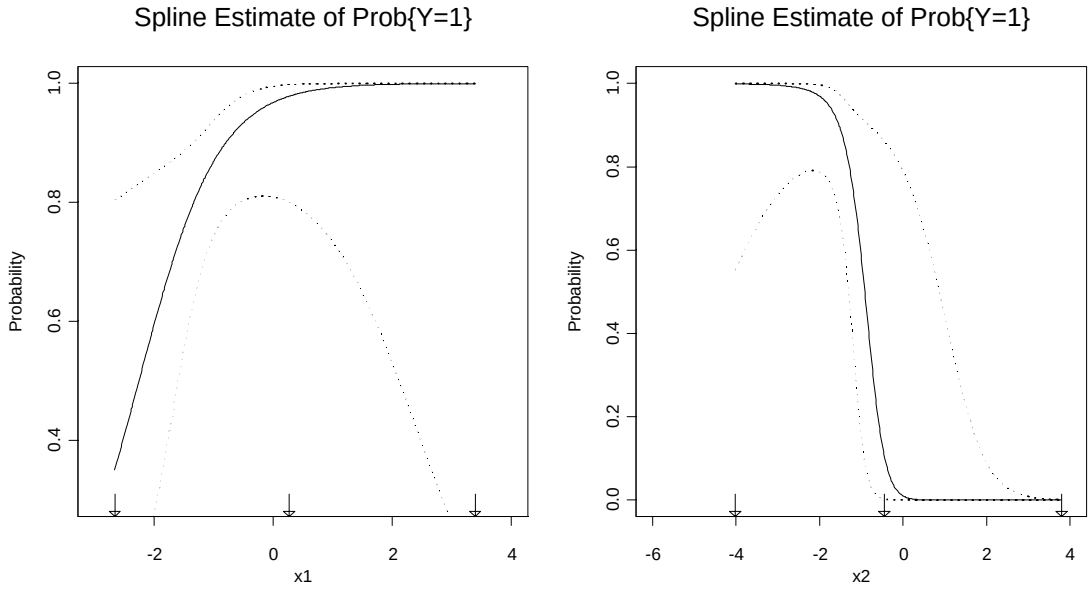
```
par(mfrow=c(1,2))
rcspline.plot(x1,y,nk=3,show="xbeta",xlab="x1",statloc="none")
rcspline.plot(x2,y,nk=3,show="xbeta",xlab="x2",statloc="none")
```



The arrows facing downward on the horizontal axis are the knots chosen by `rcs`. Alternatively, one can set them. From the looks of the confidence limits around the transformations, these are not very accurate.

Or, we could look at the effect of each splined x on the probability that y=1

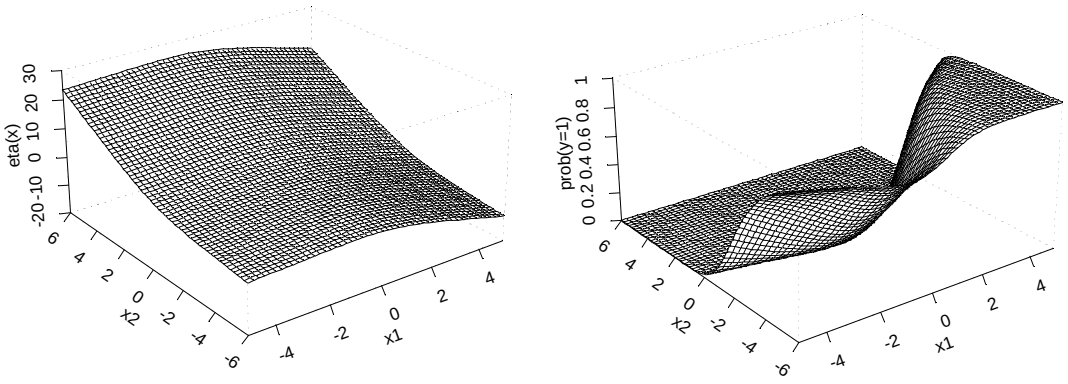
```
rcspline.plot(x1,y,nk=3,show="prob",xlab="x1",statloc="none")
rcspline.plot(x2,y,nk=3,show="prob",xlab="x2",statloc="none")
```



The joint effect of `rcs(x1)` and `rcs(x2)` is seen in the following two plots. The first plot shows the effect on the "linear" predictor, $\text{rcs}(\mathbf{X})^{\mathrm{T}} b$, the common part of each log odds $\Pr(y \geq j | \mathbf{X})$. Note that $\text{rcs}(\mathbf{X})$ represents the splined x's. The second plot shows the joint effect of `rcs(x1)` and `rcs(x2)` on $\Pr(y = 1 | \mathbf{X})$.

```
grid <- list(x1=seq(-5,5,0.2),x2=seq(-6,6,0.2))
pred <- matrix(predict(fit,expand.grid(grid),type="lp"),
            length(grid$x1))
persp(grid$x1, grid$x2, pred, xlab="x1", ylab="x2",
       zlab="eta(x)")

pred <- matrix(predict(fit,expand.grid(grid),type="fitted.ind")[,1],
            length(grid$x1))
persp(grid$x1, grid$x2, pred, xlab="x1", ylab="x2",
       zlab="prob(y=1)")
```



These plots are very similar to those obtained from `nolr`.

Chapter 10 - Agresti

A. Comparing dependent proportions

Mcnemar's test is easily conducted on the data in Table 10.1 using the built-in Splus function, `mcnemar.test`

```
mcnemar.test(matrix(c(794,150,86,570),byrow=T,ncol=2),correct=F)

   McNemar's chi-square test without continuity correction

data:  matrix(c(794, 150, 86, 570), byrow = T, ncol = 2)
McNemar's chi-square = 17.3559, df = 1, p-value = 0
```

The statistic reported is Agresti's z_0^2, which indicates a significant change in presidential approval rating. The direction of the change can, of course, be determined by computing the statistic, d, on p.348.

B. Conditional Logistic Regression (CLR)

In this section, I demonstrate how one might use CLR to fit a logistic model for binary matched pairs in Section 10.1.4 in Agresti. I use the data from problem 10.1 on p. 375. This method was originally introduced by T. Therneau and uses his `coxph` function, which comes with Splus.

Here is how I set up the data:

```
group<-rep(1:500,rep(2,500)) # each distinct number represents a pair
status<-rep(c(1,0),500)   # status indicates husband(1) or wife (0)
party<-c(rep("Demo",400),rep(c("Demo","Rep"),25),rep(c("Rep","Demo"),
   75),rep("Rep",400))
```

To fit the model, I use the function `coxph`. The "time" setting for the function `Surv` is just a vector of ones (`rep(1,1000)`) with as many values as persons.

```
fit<-coxph(Surv(rep(1,1000),status)~party+strata(group),method="exact")

Call:
coxph(formula = Surv(rep(1, 1000), status) ~ party + strata(group), method =
   "exact")

  n= 1000

      coef exp(coef) se(coef)    z      p
party  1.1         3    0.231 4.76 2e-006

      exp(coef) exp(-coef) lower .95 upper .95
party         3      0.333      1.91      4.72

Rsquare= 0.026   (max possible= 0.5 )
Likelihood ratio test= 26.2  on 1 df,   p=3.14e-007
Wald test            = 22.6  on 1 df,   p=1.96e-006
Efficient score test = 25  on 1 df,   p=5.73e-007
```

From the output, we obtain that the estimate of the odds ratio is 3. The odds of being a Democrat are three times more likely for the wife of the matched pair than for the husband. The effect of political party is significant. Thus, marginal homogeneity is rejected.

C. Symmetry Models

To fit symmetry models, Agresti uses the migration data in Table 10.2, obtained in Splus using the following commands.

```
residence80<-(c("NE","MW","S","W"))
residence85<-(c("NE","MW","S","W"))

table.10.2<-expand.grid(res80=residence80,res85=residence85,symm=c("1","2"))
table.10.2$counts<-c(11607,100,366,124,87,13677,515,302,172,225,17819,
   270,63,176,286,10192,11607,87,172,63,100,13677,225,176,366,515,
   17819,286,124,302,270,10192)
```

The "symm" factor in data frame table.10.2 is used as a result of the suggestion in problem 10.27 (p. 382), which explains how quasi-symmetry models can be fit using loglinear models. "symm" is factor "Z", with two levels.

Ordinary Symmetry Model

An ordinary symmetry model is simply fit as a two-factor log-linear model, as shown here.

```
fit<-glm(counts~res80*res85,family=poisson(link=log),data=table.10.2)

Coefficients:
 (Intercept)    res801  res802      res803    res851  res852      res853
       res801res851
    6.329792 0.1776753 0.18193 -0.03731356 0.1776753 0.18193 -0.03731356
   2.451728

 res802res851 res803res851 res801res852 res802res852 res803res852 res801res853
   0.03945671   0.09719178   0.03945671    0.5854987  0.008290874   0.09719178

 res802res853 res803res853
  0.008290874    0.3470497

Degrees of Freedom: 32 Total; 16 Residual
Residual Deviance: 487.1004
```

As it says in problem 10.27, we need to divide the Residual Deviance by 2 to get the true Goodness of Fit statistic:

```
fit$deviance/2
[1] 243.5502
```

It is also apparent that one must compute the df by hand since the residual df for the symmetry model is 6, not 16, and the total is half of 32.

Another way to fit the symmetry model, which does *not* require the adjustment of df or of GoF statistics, is via a method proposed by A. Zaslavsky. The data is set up in the same way, with the exception of the "symm" factor.

```
table.10.2<-expand.grid(res80=residence80,res85=residence85)
table.10.2$counts<-
   c(11607,100,366,124,87,13677,515,302,172,225,17819,270,63,176,286,10192)
table.10.2$symm<-paste(
   pmin(as.numeric(table.10.2$res80),as.numeric(table.10.2$res85)),
   pmax(as.numeric(table.10.2$res80),as.numeric(table.10.2$res85)),sep=",")
   res80 res85 counts symm
 1    NE    NE  11607  1,1
 2    MW    NE    100  1,2
```

```
 3     S    NE    366  1,3
 4     W    NE    124  1,4
 5    NE    MW     87  1,2
 6    MW    MW  13677  2,2
 7     S    MW    515  2,3
 8     W    MW    302  2,4
 9    NE     S    172  1,3
10    MW     S    225  2,3
11     S     S  17819  3,3
12     W     S    270  3,4
13    NE     W     63  1,4
14    MW     W    176  2,4
15     S     W    286  3,4
16     W     W  10192  4,4
```

To fit the symmetry model, just use “symm” as the factor:

```
fit<-glm(counts~symm,family=poisson(log),data=table.10.2)
```

```
Coefficients:
 (Intercept)     symm1      symm2      symm3     symm4      symm5     symm6
       symm7
    6.958844 -2.410701 -0.4513163 -0.4898459 0.7031941 -0.1328645 -0.157338
   0.4209411
     symm8     symm9
 -0.134868 0.2522793

Degrees of Freedom: 16 Total; 6 Residual
Residual Deviance: 243.5502
```

Note that one does not need to adjust the Goodness of Fit statistic nor the df.

Quasi-Symmetry Model

Using the suggestion in problem 10.27, and the first definition of “symm”, we have the quasi-symmetry fit:

```
fit<-glm(counts~symm*res80+symm*res85+res80*res85,family=poisson(link=log),
   data=table.10.2)
```

```
Coefficients:
 (Intercept)         symm    res801    res802      res803    res851    res852
    6.298585 1.77305e-016 0.1797539 0.1800794 -0.03565628 0.1797539 0.1800794

      res853  symmres801 symmres802  symmres803 symmres851 symmres852 symmres853
 -0.03565628 -0.01224104  -0.128141 -0.04887195 0.01224104   0.128141 0.04887195

 res801res851 res802res851 res803res851 res801res852 res802res852 res803res852
     2.451878   0.04095189   0.09777525   0.04095189    0.6015282   0.01442996

 res801res853 res802res853 res803res853
   0.09777525   0.01442996    0.3494123

Degrees of Freedom: 32 Total; 9 Residual
Residual Deviance: 5.971925
```

```
fit$deviance/2
```
```
[1] 2.985962
```

Fitted values for the quasi-symmetry model are obtained as follows:

```
matrix(fitted(fit2),ncol=4,byrow=T,dimnames=list(rep(residence80,2),
   residence85))[1:4,] # we take only the first set of fitted values
```

```
            NE          MW          S          W
NE 11607.00000    95.78862   370.4375   123.7739
MW    91.21138 13677.00000   501.6825   311.1061
 S   167.56253   238.31746 17819.0000   261.1200
 W    63.22609   166.89392   294.8800 10192.0000
```

Using Zaslavsky's definition of "symm", a fit of the quasi-symmetry model is obtained using

```
fit<-glm(counts~symm+res80,family=poisson(link=log),data=table.10.2)
```

```
Coefficients:
 (Intercept)     symm1      symm2      symm3     symm4      symm5      symm6
       symm7
    6.933879 -2.423092 -0.6049717 -0.5461458 0.7378705 -0.1830642 -0.1816523
   0.354548

      symm8    symm9     res801   res802    res803
 -0.1799573 0.222472 0.02448208 0.256282 0.0977439


Degrees of Freedom: 16 Total; 3 Residual
Residual Deviance: 2.985962
```

To obtain fitted values, type:

```
matrix(fitted(fit2),ncol=4,byrow=T,dimnames=list(residence80, residence85))[1:4,] #
   we take only the first set of fitted values
```

```
            NE          MW          S          W
NE 11607.00000    95.78862   370.4375   123.7739
MW    91.21138 13677.00000   501.6825   311.1061
 S   167.56253   238.31746 17819.0000   261.1200
 W    63.22609   166.89392   294.8800 10192.0000
```

Quasi-independence Model

The quasi-independence model is fit in two ways. First, I use the method in GLIM described in Table A.20 of Appendix A of Agresti. Then, I use the method of SAS PROC CATMOD, also mentioned in Appendix A.

The method in GLIM adds to the independence model four indicator variables that represent the main diagonal cells.

```
table.10.2<-expand.grid(res80=residence80,res85=residence85)
table.10.2$counts<-
   c(11607,100,366,124,87,13677,515,302,172,225,17819,270,63,176,286,10192)
```

I use these Indicator variables:

```
table.10.2$D1<-c(1,rep(0,15))
table.10.2$D2<-c(rep(0,5),1,rep(0,10))
table.10.2$D3<-c(rep(0,10),1,rep(0,5))
table.10.2$D4<-c(rep(0,15),1)
```

Now, fit the quasi-independence model:

```
fit<-glm(counts~res80+res85+D1+D2+D3+D4,family=poisson(link=log),
   data=table.10.2)
```

```
Degrees of Freedom: 16 Total; 5 Residual
Residual Deviance: 69.5094
```

The fitted values for the quasi-independence model are obtained as follows:

```
matrix(fitted(fit),ncol=4,byrow=T,dimnames=list(residence80, residence85))[1:4,] #
   we take only the first set of fitted values

           NE         MW          S          W
NE 11607.0000   126.5622   312.9086   150.5292
MW   117.3881 13677.0000   531.1125   255.4994
 S   133.2261   243.8025 17819.0000   289.9714
 W    71.3858   130.6353   322.9789 10192.0000
```

Appendix A in Agresti (p. 500) also suggests fitting the quasi-independence model as SAS PROC CATMOD does. That is, zero-out the main diagonal of the contingency table, and fit an independence log-linear model, treating the main diagonal zeroes as structural zeroes. I do that here using `loglin` with the 'start' option.

```
table.10.2<-expand.grid(res80=residence80,res85=residence85)
table.10.2$counts<-c(0,100,366,124,87,0,515,302,172,225,0,270,63,176,286,0) # note
   the zeroes
table.10.2<-t(design.table(table.10.2)) # create a table out of the data frame
loglin(table.10.2,margin=list(1,2),start=abs(as.numeric(diag(4))-1),fit=T)

7 iterations: deviation 0.0811768
$lrt:
[1] 69.50941

$pearson:
[1] 71.15848

$df:
[1] 9

$margin:
$margin[[1]]:
[1] "res80"

$margin[[2]]:
[1] "res85"


$fit:
          NE       MW        S        W
NE   0.00000 126.5636 312.9075 150.5306
MW 117.38965   0.0000 531.1134 255.5030
 S 133.22371 243.7990   0.0000 289.9664
 W  71.38664 130.6374 322.9791   0.0000
```

As noted in a previous section, when we have structural zeroes in `loglin`, we must adjust the df for the zeroes. Thus, we can subtract 4 to get df = 5.

D. Marginal Homogeneity Models

In this section, the marginal homogeneity model is fit to the migration data using the model matrix in Table A.19 of Appendix A in Agresti.

First, I set up the dummy matrix in Table A.19 as a matrix, then I make it a data frame:

```
dummies<-matrix(0,nr=16,nc=13)
dummies[cbind(c(1:3,5:7,9:11),c(1:3,4:6,7:9))]<-1
dummies[cbind(rep(c(4,8,12),rep(3,3)),c(1:3,4:6,7:9))]<--1
dummies[cbind(rep(13:15,3),c(1:3,4:6,7:9))]<--1
dummies[16,10]<-1
diag(dummies[c(4,8,12),c(11,12,13)])<-1
```

```
diag(dummies[c(13,14,15),c(11,12,13)])<-1

dummies<-data.frame(counts=c(11607,100,366,124,87,13677,515,302,172,
   225,17819,270,63,176,286,10192),dummies)
names(dummies)<-c("counts","m11","m12","m13","m21","m22","m23","m31",
   "m32","m33","m44","m1","m2","m3")
```

```
   counts m11 m12 m13 m21 m22 m23 m31 m32 m33 m44 m1 m2 m3
 1  11607   1   0   0   0   0   0   0   0   0   0  0  0  0
 2    100   0   1   0   0   0   0   0   0   0   0  0  0  0
 3    366   0   0   1   0   0   0   0   0   0   0  0  0  0
 4    124  -1  -1  -1   0   0   0   0   0   0   0  1  0  0
 5     87   0   0   0   1   0   0   0   0   0   0  0  0  0
 6  13677   0   0   0   0   1   0   0   0   0   0  0  0  0
 7    515   0   0   0   0   0   1   0   0   0   0  0  0  0
 8    302   0   0   0  -1  -1  -1   0   0   0   0  0  1  0
 9    172   0   0   0   0   0   0   1   0   0   0  0  0  0
10    225   0   0   0   0   0   0   0   1   0   0  0  0  0
11  17819   0   0   0   0   0   0   0   0   1   0  0  0  0
12    270   0   0   0   0   0   0  -1  -1  -1   0  0  0  1
13     63  -1   0   0  -1   0   0  -1   0   0   0  1  0  0
14    176   0  -1   0   0  -1   0   0  -1   0   0  0  1  0
15    286   0   0  -1   0   0  -1   0   0  -1   0  0  0  1
16  10192   0   0   0   0   0   0   0   0   0   1  0  0  0
```

The above matrix is supposed to encompass the constraints on the expected cell entries.

Now, I fit the marginal homogeneity model using a poisson family with identity link:

```
fit<-glm(counts~.-1,family=poisson(identity),data=dummies)

Degrees of Freedom: 16 Total; 3 Residual
Residual Deviance: 240.7458

matrix(fitted(fit),byrow=T,nc=4,dimnames=list(residence80,residence85))
```

```
            NE          MW          S           W
NE 11607.00000    98.08285   265.6867    93.96975
MW    88.73298 13677.00000   379.0733   232.34523
 S   276.47479   350.80060 17819.0000   287.25936
 W    92.53157   251.26811   269.7747 10192.00000
```

Check that the row and column sums match:

```
rowSums(matrix(fitted(fit),byrow=T,nc=4))
[1] 12064.74 14377.15 18733.53 10805.57

colSums(matrix(fitted(fit),byrow=T,nc=4))
[1] 12064.74 14377.15 18733.53 10805.57
```

Another way to test marginal homogeneity is to use Bhapkar's (1966) and Stuart's (1955) asymptotic test. However, instead of using Lipsitz's (1988) matrix, we can use a matrix that is more transparent in terms of how easily it is to see the constraints. The development is found in Wickens (1989).

The constraints that all cross marginal totals are equal is equivalent to the following three equations

$$p_{12} + p_{13} + p_{14} = p_{21} + p_{31} + p_{41}$$

$$p_{12} + p_{32} + p_{42} = p_{21} + p_{23} + p_{24}$$

$$p_{13} + p_{23} + p_{43} = p_{31} + p_{32} + p_{34}$$

which we can put into a contrast matrix directly:

```
A<-matrix(c(0,1,1,1,-1,0,0,0,-1,0,0,0,-1,0,0,0,
            0,1,0,0,-1,0,-1,-1,0,1,0,0,0,1,0,0,
            0,0,1,0,0,0,1,0,-1,-1,0,-1,0,0,1,0),nc=16,nr=3,byrow=T)
```

Now, get the sample proportions and compute the linear combinations imposed by the constraint matrix, A.

```
p<-(dummies$counts/sum(dummies$counts))
y<-A%*%p
```

Now, get Sp, the var/cov matrix of p. (This is a complicated way to do it. I'm sure there is a simpler way.)

```
Sp<-diag(p*(1-p))
oldres<-kronecker(p[1],p[-1])
for(i in 2:15){ res<-kronecker(p[i],p[-(1:i)]); res<-c(oldres,res); oldres<-res }
Sp[lower.tri(Sp)]<--res
temp<-t(Sp)
temp[lower.tri(Sp)]<--res
Sp<-temp/sum(dummies$counts)
```

Now, get Sy, the covariance matrix of the linear combinations:

```
Sy<-A%*%Sp%*%t(A)
```

And, compute W:

```
as.numeric(t(y)%*%solve(Sy)%*%y)
[1] 236.4906
```

E. Square Tables with Ordered Categories

Using the data in Table 10.4, I fit the symmetry and conditional symmetry models.

```
table.10.4<-data.frame(expand.grid(case=c(0,1,2,3),control=c(0,1,2,3)),
   counts=c(31,12,14,6,5,1,1,1,5,0,2,1,0,0,1,0))
```

Symmetry model

I modify table.10.4 by adding a symmetry factor:

```
table.10.4$symm<-paste(
   pmin(as.numeric(table.10.4$case),as.numeric(table.10.4$control)),
   pmax(as.numeric(table.10.4$case),as.numeric(table.10.4$control)),sep=",")
```

and obtain the chi-squared statistic using `.pearson.x2`

```
fit<-glm(counts~symm,data=table.10.4,family=poisson(log))
.pearson.x2(observed=table.10.4$counts,expected=fitted(fit))$X2
[1] 15.14592
```

Conditional Symmetry Model

To fit a conditional symmetry model, I first obtain a vector that represents tau:

```
temp<-matrix(0,nr=4,nc=4)
tau<-as.numeric(ifelse(row(temp)<col(temp),1,0))
```

```
[1] 0 0 0 0 1 0 0 0 1 1 0 0 1 1 1 0
```

Then, I add tau as a factor in the model:

```
fit<-glm(counts~symm+tau,family=poisson(log),data=table.10.4)

Coefficients:
 (Intercept)      symm1      symm2      symm3      symm4      symm5      symm6
   0.1960172 -0.4370336 -0.1086027 -0.3424712 -0.5091759 -0.3849995 -0.2749997

       symm7      symm8      symm9       tau
 -0.08544468 -0.0968229 -0.9984199 -1.157453

Degrees of Freedom: 16 Total; 5 Residual
Residual Deviance: 5.336852
```

The model has a chi-squared GOF value:

```
.pearson.x2(observed=table.10.4$counts,expected=fitted(fit))$X2
[1] 3.605283
```

To obtain the fitted values for the conditional symmetry model, one may type:

```
matrix(round(fitted(fit),1),nr=4,nc=4,byrow=F,dimnames=list(case=c(0,1,2,3),control=
   c(0,1,2,3)))

     0   1   2   3
0 31.0 4.1 4.5 1.4
1 12.9 1.0 0.2 0.2
2 14.5 0.8 2.0 0.5
3  4.6 0.8 1.5 0.0
```

F. Measuring Agreement

For this section, I use the Pathologist data in Table 10.5. Here is how I set it up.

```
PathA<-factor(c(1,2,3,4,5))
PathB<-PathA
table.10.5<-data.frame(expand.grid(PathA=PathA,PathB=PathB),
   count=c(22,5,0,0,0,2,7,2,1,0,2,14,36,14,3,0,0,0,7,0,0,0,0,0,3))
table.10.5<-design.table(table.10.5)
```

Cohen's Kappa

One way to compute Kappa is with the following function:

```
Kappa<-function(table){
   pij<-table/sum(table)
   pio<-sum(diag(pij))
   pie<-sum(rowSums(pij)*colSums(pij))
   (pio-pie)/(1-pie)
}
```

Applied to Table 10.5, we get

```
Kappa(table.10.5)
[1] 0.4984183
```

Uniform Association Plus Extra Agreement Model

To fit a linear-by-linear association plus extra agreement model (10.28) to Table 10.5, I will use unit-spaced scores.

```
delta<-as.numeric(diag(5))# Gives 1 for diagonal entries, 0 otherwise
table.10.5<-data.frame(expand.grid(PathA=factor(PathA),PathB=factor(PathB)),
   count=c(22,5,0,0,0,2,7,2,1,0,2,14,36,14,3,0,0,0,7,0,0,0,0,0,3))
fit<-glm(count~PathA+PathB+delta+codes(PathA):codes(PathB),
   family=poisson(log),data=table.10.5)   # The codes function gives the unit-spaced
   scores.

summary(fit,cor=F)

Coefficients:
                              Value Std. Error   t value
             (Intercept) -11.4248226  3.5087037 -3.256138
                  PathA1  -0.8506102  0.3803249 -2.236536
                  PathA2  -1.4361990  0.3928591 -3.655761
                  PathA3  -1.5228694  0.5070563 -3.003353
                  PathA4  -1.9647665  0.5572738 -3.525675
                  PathB1  -1.3273248  0.3009011 -4.411166
                  PathB2  -0.7410464  0.3822727 -1.938528
                  PathB3  -2.0525682  0.4964779 -4.134259
                  PathB4  -2.4092934  0.6539193 -3.684389
                   delta   1.0668241  0.4037553  2.642254
codes(PathA):codes(PathB)   1.1498870  0.3418667  3.363554

(Dispersion Parameter for Poisson family taken to be 1 )

    Null Deviance: 267.6605 on 24 degrees of freedom
Residual Deviance: 8.411956 on 14 degrees of freedom
```

The underlined values in the list of coefficients are estimates of d and b, respectively, in Agresti.

To obtain a matrix of the fitted values, one may type:

```
matrix(round(fitted(fit),1),nr=5,nc=5,dimnames=list(PathA,PathB))

     1   2    3   4   5
1 22.1 1.7  2.2 0.0 0.0
2  4.4 8.9 12.5 0.1 0.0
3  0.4 1.0 36.1 0.5 0.0
4  0.1 0.4 15.6 5.4 0.5
5  0.0 0.0  2.6 1.0 2.4
```

G. Bradley-Terry Model

We fit the Bradley-Terry model for paired comparisons using the data in Table 10.6. There are several ways to fit this model, given a set of paired comparisons. I will fit it first as a quasi-symmetry model, then as a logit model.

To set up the data, type

```
losing.team<-c("Milwaukee","Detroit","Toronto","NY","Boston","Cleveland",
   "Baltimore")
win.team<-losing.team

table.10.6<-expand.grid(losing=factor(losing.team),winning=factor(win.team))
table.10.6$counts<-c(0,7,9,7,7,9,11,6,0,7,5,11,9,9,4,6,0,7,7,8,12,6,8,
   6,0,6,7,10,6,2, 6,7,0,7,12,4,4,5,6,6,0,6,2,4,1,3,1,7,0)
```

```
table.10.6$symm<-paste( pmin(as.numeric(factor(table.10.6$winning,levels=win.team)),
   as.numeric(factor(table.10.6$losing,levels=losing.team))),
   pmax(as.numeric(factor(table.10.6$winning,levels=win.team)),
   as.numeric(factor(table.10.6$losing,levels=losing.team))),sep=",")
```

Quasi-symmetry fit:

```
fit<-glm(counts~symm+losing,data=table.10.6,family=poisson(log))

Degrees of Freedom: 49 Total; 15 Residual
Residual Deviance: 15.73729
```

Fitted values are obtained by

```
matrix(round(fitted(fit),1),nr=7,nc=7,byrow=T,dimnames=list(win.team, losing.team))
   # byrow=T matches the data entry of the counts above
```

```
          Milwaukee Detroit Toronto  NY Boston Cleveland Baltimore
Milwaukee       0.0     7.0     7.4 7.6    8.0       9.2      10.8
  Detroit       6.0     0.0     7.0 7.1    7.6       8.8      10.5
  Toronto       5.6     6.0     0.0 6.7    7.1       8.4      10.2
       NY       5.4     5.9     6.3 0.0    7.0       8.3      10.1
   Boston       5.0     5.4     5.9 6.0    0.0       7.9       9.8
Cleveland       3.8     4.2     4.6 4.7    5.1       0.0       8.6
Baltimore       2.2     2.5     2.8 2.9    3.2       4.4       0.0
```

Logit model fit (see Appendix A in Agresti):

Appendix A states that one can fit a Bradley-Terry model by creating seven artificial variables, corresponding to the seven teams, and then fitting a logit model to the counts of wins. Here, X1 - X7 are the artificial variables, which equal 1 if the particular team won the game represented by the cell, -1 if the team lost the game, and 0 if the team did not play.

```
X1<-c(-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
X2<-c(1,0,0,0,0,0,-1,-1,-1,-1,-1,0,0,0,0,0,0,0,0,0,0)
X3<-c(0,1,0,0,0,0,1,0,0,0,0,-1,-1,-1,-1,0,0,0,0,0,0)
X4<-c(0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,-1,-1,-1,0,0,0)
X5<-c(0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,1,0,0,-1,-1,0)
X6<-c(0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,1,0,1,0,-1)
X7<-c(0,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,1,0,1,0)

response<-cbind(c(6,4,6,6,4,2,6,8,2,4,4,6,6,5,1,7,6,3,6,1,7), 13-
   c(6,4,6,6,4,2,6,8,2,4,4,6,6,5,1,7,6,3,6,1,7))
```

The first column of “response” is the number of wins in each of the 21 games; the second is the number of losses.

```
      [,1] [,2]
 [1,]    6    7
 [2,]    4    9
 [3,]    6    7
 [4,]    6    7
 [5,]    4    9
 [6,]    2   11
 [7,]    6    7
 [8,]    8    5
 [9,]    2   11
[10,]    4    9
[11,]    4    9
[12,]    6    7
[13,]    6    7
[14,]    5    8
[15,]    1   12
```

```
[16,]    7    6
[17,]    6    7
[18,]    3   10
[19,]    6    7
[20,]    1   12
[21,]    7    6
```

Here is the fit of a logit model:

```
fit<-glm(response~-1+X1+X2+X3+X4+X5+X6, family=binomial)   # exclude intercept

Degrees of Freedom: 21 Total; 15 Residual
Residual Deviance: 15.7365
```

and its fitted values:

```
junk<-matrix(0,nc=7,nr=7,dimnames=list(win.team,win.team))
junk[lower.tri(junk)]<-round(13*fitted(fit),1)
junk<-t(junk)
junk[lower.tri(junk)]<-13-round(13*fitted(fit),1)
junk<-t(junk)
```

	Milwaukee	Detroit	Toronto	NY	Boston	Cleveland	Baltimore
Milwaukee	0.0	7.0	7.4	7.6	8.0	9.2	10.8
Detroit	6.0	0.0	7.0	7.1	7.6	8.8	10.5
Toronto	5.6	6.0	0.0	6.7	7.1	8.4	10.2
NY	5.4	5.9	6.3	0.0	7.0	8.3	10.1
Boston	5.0	5.4	5.9	6.0	0.0	7.9	9.8
Cleveland	3.8	4.2	4.6	4.7	5.1	0.0	8.6
Baltimore	2.2	2.5	2.8	2.9	3.2	4.4	0.0

Issuing the command `fitted(fit)` gives the Π_{ij}'s.

H. Bradley-Terry Model-with Order Effect (rough draft)

Apparently, this model can be fit in many ways. Of all the ways I tried, the logit model fit (as per p. 373, Agresti) gave the closest answers. However, I still can't get this to work.

As mentioned in Agresti, the 42 pair sets (where order matters) can be viewed as 42 independent binomial samples, where response is the number of successes (wins) out of the total played.

```
response<-cbind(c(4,4,4,6,4,6,4,4,6,6,4,2,4,4,6,4,4,6,5,6,2,
                  3,2,3,5,2,2,4,5,2,3,1,2,3,3,1,4,4,2,4,1,3),
                c(3,2,3,1,2,0,2,3,0,1,3,4,3,2,0,3,2,1,2,0,4,
                  3,5,3,1,5,5,3,1,5,3,5,5,3,4,6,2,3,4,2,6,4) )
```

I create the "home advantage" variable by giving each of the 42 game sets a 1 if a home team won the set and 0 if an away team won or if there was a tie in the number of games won.

```
Home<-c(1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,
        0,0,0,1,0,0,1,1,0,0,0,0,0,0,0,1,1,0,1,0,0)  # ties give no advantage
```

Then, the seven artificial variables (similar to those above, except now there are 42) are set up as follows, with a 1 for a win by that team, -1 for a loss, and 0 for no-play. There are several ways I could have set up ties; but no way gave correct answers in the model fit.

```
X1<-c( 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  0.0, 1,  0.0,-1, 1, 1, 0, 0, 0,  0,   0, 0,  0,   0, 0, 0, 0, 0, 0, 0, 0)
X2<-c(-1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  0.0, 0,  0,   0, 0, 0,-1,-1, 1,  0.0, 1, 0,  0,   0, 0, 0, 0, 0, 0, 0, 0)
X3<-c( 0,-1, 0, 0, 0, 0,-1, 0, 0, 0, 0,-1, 1, 1, 1, 0, 0, 0, 0, 0, 0,  0,  -1,  0,   0, 0, 0, 1, 0, 0,  0,   0, 1,  0.0, 1, 1, 0, 0, 0, 0, 0, 0)
X4<-c( 0, 0,-1, 0, 0, 0, 0,-1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,  0,   0,  0.0, 0, 0, 0, 0, 1, 0,  0,   0,-1,  0,   0, 0,-1,-1, 1, 0, 0, 0)
X5<-c( 0, 0, 0,-1, 0, 0, 0, 0,-1, 0, 0, 0,-1, 0, 0,-1, 0, 0, 1, 1, 0,  0,   0,  0,   1, 0, 0, 0, 0,-1,  0,   0, 0,  0.0, 0, 0, 1, 0, 0,-1, 1, 0)
X6<-c( 0, 0, 0, 0,-1, 0, 0, 0, 0, 0,-1, 0, 0, 0,-1, 0, 0,-1, 0,-1, 0,  0,   0,  0,   0,-1, 0, 0, 0, 0,  0.0, 0, 0,  0,  -1, 0, 0, 1, 0, 1, 0, 1)
X7<-c( 0, 0, 0, 0, 0, 0,-1, 0, 0, 0, 0, 0,-1, 0, 0, 0,-1, 0, 0,-1, 0,-1, 1,  0,   0,  0,   0, 0, 0,-1, 0, 0, 0, 0,  0,  -1, 0,  0,   0,-1, 0, 0,-1, 0,-1,-1)
```

Model fit:

glm(response~-1+X1+X2+X3+X4+X5+X6+Home,family=binomial)

```
Degrees of Freedom: 42 Total; 35 Residual
Residual Deviance: 40.38968
```

Chapter 11 - Agresti

A. Symmetry Models for Repeated Categorical Responses

Agresti fits two symmetry models to the data in Table 11.1 (Attitudes towards legalized abortion): Complete symmetry and Quasi-symmetry.

First, set up the data

```
table.11.1<-data.frame(expand.grid(Poor=c("y","n"),Unmarried=c("y","n"),
   Rape=c("y","n"),Defect=c("y","n")), counts=c(605,68,91,320,1,0,3,45,7,
   3,7,54,2,0,3,125))
```

Complete Symmetry

To fit a complete symmetry model to Table 11.1, I use the suggestion by A. Zaslavsky, but generalize it to a model with T=4 factors (conditions) and I=2 responses under each factor.

Here is the resulting table. Note that I label “symm” as "4" for all rows (subject profiles) with 4 yeses. I give “symm” the label "3" for all rows with 3 yeses, etc. Thus, the complete symmetry model has 5 parameters. This model assumes that all orders of the responses are equal.

```
table.11.1$symm<-as.factor(rowSums(cbind(table.11.1$Poor,table.11.1$Unmarried,
   table.11.1$Rape,table.11.1$Defect)-1))
```

```
table.11.1
   Poor Unmarried Rape Defect counts symm
 1    y         y    y      y    605    0
 2    n         y    y      y     68    1
 3    y         n    y      y     91    1
 4    n         n    y      y    320    2
 5    y         y    n      y      1    1
 6    n         y    n      y      0    2
 7    y         n    n      y      3    2
 8    n         n    n      y     45    3
 9    y         y    y      n      7    1
10    n         y    y      n      3    2
11    y         n    y      n      7    2
12    n         n    y      n     54    3
13    y         y    n      n      2    2
14    n         y    n      n      0    3
15    y         n    n      n      3    3
16    n         n    n      n    125    4
```

Now, I fit the model:

```
fit<-glm(counts~symm,data=table.11.1,family=poisson(log))
```

```
Coefficients:
 (Intercept)     symm1      symm2     symm3      symm4
    4.445258 -1.336765 -0.3486976 -0.370272 0.09576387

Degrees of Freedom: 16 Total; 11 Residual
Residual Deviance: 1334.931
```

Quasi-symmetry Model

Generalizing the same method as was used in Chapter 10 to a higher dimensional table, we get the fit,

```
fit<-glm(counts~symm+Unmarried+Rape+Defect,family=poisson(link=log),
   data=table.11.1)
```

```
Degrees of Freedom: 16 Total; 8 Residual
Residual Deviance: 33.23141
```

Note that I have excluded the “Poor” variable from the model formula above. If I had kept all variables in the formula, the last mentioned variable would not be estimated.

No-three-factor interaction model

Agresti compares the complete symmetry model and quasi-symmetry model to a No-three-factor interaction model and finds large differences in the estimated conditional odds ratios for the latter model.

Here is the model fit.

```
fit<-glm(counts~(Unmarried+Rape+Poor+Defect)^2,family=poisson(link=log),
   data=table.11.1)
```

```
Coefficients:
 (Intercept) Unmarried      Rape      Poor     Defect Unmarried:Rape
    2.439943 0.9590131 -1.278875 0.3275534 -0.7938695      0.6603908

 Unmarried:Poor Unmarried:Defect Rape:Poor Rape:Defect Poor:Defect
      0.8637749        0.3821812 0.3306227   0.7118181   0.2291614

Degrees of Freedom: 16 Total; 5 Residual
Residual Deviance: 6.567354
```

And, the conditional odds ratios can be obtained using, for example,

```
exp(4*fit$coefficients[10])
```

```
 Rape:Defect
    17.24069
```

for the DR odds ratio.

B. Marginal Homogeneity

Marginal homogeneity implies T identical response distributions across T time periods. Agresti tests marginal homogeneity of the binary response distributions (success/failure) across three drugs for the data in Table 11.2.

```
table.11.2<-data.frame(expand.grid(C=c("y","n"),B=c("y","n"),A=c("y","n")),
      counts=c(6,16,2,4,2,4,6,6))
```

```
  C B A counts
1 y y y      6
2 n y y     16
3 y n y      2
4 n n y      4
5 y y n      2
6 n y n      4
7 y n n      6
8 n n n      6
```

First, I compute Bhapkar's W statistic, following the same logic (i.e., from Wickens, 1989)as in Chapter 10.

The constraint that all marginals (A, B, C) are equal is equivalent to the following two equations.

```
A<-matrix(c(0,0,1,1,-1,-1,0,0,
            0,1,-1,0,0,1,-1,0),nc=8,nr=2,byrow=T)
```

Then, the following commands will calculate W:

```
prob<-(table.11.2$counts/sum(table.11.2$counts))
y<-A%*%prob

Sp<-diag(prob*(1-prob))
oldres<-kronecker(prob[1],prob[-1])
for(i in 2:7){ res<-kronecker(prob[i],prob[-(1:i)]); res<-c(oldres,res); oldres<-res
   }
Sp[lower.tri(Sp)]<--res
temp<-t(Sp)
temp[lower.tri(Sp)]<--res
Sp<-temp/sum(table.11.2$counts)

Sy<-A%*%Sp%*%t(A)
```

Compute W:

```
as.numeric(t(y)%*%solve(Sy)%*%y)
[1] 6.584493
```

For Madansky's LRT, we can use the equations already set up in Table A.21 in Agresti for each of the eight expected frequencies, subject to the constraint of marginal homogeneity.

```
dummies<-matrix(c(1,0,0,0,0,0,0,0,
          -1,1,0,0,0,0,
          -1,0,1,0,0,0,
           1,-1,-1,0,1,0,
          -1,0,0,1,0,0,
           1,-1,0,-1,1,0,
           1,0,-1,-1,1,0,
           0,0,0,0,0,1),byrow=T,nr=8)

table.11.2<-data.frame(counts=table.11.2$counts,dummies)
names(table.11.2)<-c("counts","m111","m11p","m1p1","mp11","m1pp","m222")
```

```
  counts m111 m11p m1p1 mp11 m1pp m222
1      6    1    0    0    0    0    0
2     16   -1    1    0    0    0    0
3      2   -1    0    1    0    0    0
4      4    1   -1   -1    0    1    0
5      2   -1    0    0    1    0    0
6      4    1   -1    0   -1    1    0
7      6    1    0   -1   -1    1    0
8      6    0    0    0    0    0    1
```

Now, perform the fit including all of the dummy variables (the '.' in the formula stands for m111 + m11p + m1p1 + mp11 + m1pp + m222) and excluding an intercept:

```
fit<-glm(counts~.-1,family=poisson(identity),data=table.11.2)
```

```
Degrees of Freedom: 8 Total; 2 Residual
Residual Deviance: 5.945079
```

Cochran's Q-statistic

If we had the individual responses, we could calculate Cochran's Q statistic (or the CMH statistic) by setting up n 3x2 matrices like that in table 11.3 as a 3x2xn array, which is then directly sent into `mantelhaen.test`.

C. Modeling a Repeated Categorical Response

I will use the functions in two Splus libraries to analyze the data in Table 11.4 in Agresti: Rmtools and YAGS. Both are used expressly for repeated categorical (and continuous, Gaussian) responses. Each requires the data to be set up differently.

RMTOOLS

Table 11.4 displays data from a repeated *binary* response (normal/abnormal suffering from mental depression).

```
table.11.4<-expand.grid(Treatment=c("standard","new"), diagnosis=c("mild","severe"),
   type=c("NNN","NNA","NAN","NAA","ANN","ANA","AAN","AAA"))
```

In the data frame below, y1, y2, and y3 represent the 0/1 responses at each occasion. The variable, "type", is actually unnecessary.

```
temp<-matrix(c(c(0,0,0),c(0,0,1),c(0,1,0),c(0,1,1),c(1,0,0),c(1,0,1),c(1,1,0),
   c(1,1,1)), nc=3, byrow=T)
temp<-temp[rep(1:nrow(temp),rep(4,nrow(temp))),]
table.11.4<-cbind.data.frame(table.11.4,temp)
table.11.4<-cbind.data.frame(table.11.4,
   counts=c(16,31,2,7,13,0,2,2,9,6,8,5,3,0,9,2,14,22,9,31,4,2,15,5,15,9,27,32,
   6,0,28,6))
names(table.11.4)[4:6]<-c("y1","y2","y3")
```

```
   Treatment diagnosis type y1 y2 y3 counts
 1  standard      mild  NNN  0  0  0     16
 2       new      mild  NNN  0  0  0     31
 3  standard    severe  NNN  0  0  0      2
 4       new    severe  NNN  0  0  0      7
 5  standard      mild  NNA  0  0  1     13
 6       new      mild  NNA  0  0  1      0
 7  standard    severe  NNA  0  0  1      2
 8       new    severe  NNA  0  0  1      2
 9  standard      mild  NAN  0  1  0      9
10       new      mild  NAN  0  1  0      6
11  standard    severe  NAN  0  1  0      8
12       new    severe  NAN  0  1  0      5
13  standard      mild  NAA  0  1  1      3
14       new      mild  NAA  0  1  1      0
15  standard    severe  NAA  0  1  1      9
16       new    severe  NAA  0  1  1      2
17  standard      mild  ANN  1  0  0     14
18       new      mild  ANN  1  0  0     22
19  standard    severe  ANN  1  0  0      9
20       new    severe  ANN  1  0  0     31
21  standard      mild  ANA  1  0  1      4
22       new      mild  ANA  1  0  1      2
23  standard    severe  ANA  1  0  1     15
24       new    severe  ANA  1  0  1      5
25  standard      mild  AAN  1  1  0     15
26       new      mild  AAN  1  1  0      9
27  standard    severe  AAN  1  1  0     27
```

```
28       new    severe  AAN  1  1  0     32
29  standard      mild  AAA  1  1  1      6
30       new      mild  AAA  1  1  1      0
31  standard    severe  AAA  1  1  1     28
32       new    severe  AAA  1  1  1      6
```

```
library(rmtools)
```

To use `rm.fit`, we must put the data frame into list format, with the counts labeled as "counts" in the list, and the time increments correctly specified. First, I designate the time increments as 1, 2, and 4 weeks. Then, I construct the list.

```
time.vec<-c(1,2,4)
table.11.4<-list(y=as.matrix(table.11.4[,4:6]), treatment=factor(table.11.4[,1],
   levels=c("new","standard")),diagnosis=factor(table.11.4[,2],
   levels=c("severe","mild")), time=time.vec, counts=table.11.4[,7])
```

Below is the fit for the first model on p. 397. This model contains main effects only. (In the formula specification, I use 1-y as the response variable because of the way I coded an N and A above. I want to model the probability of a *normal* response, not an abnormal response.

```
options("contrasts"=c("contr.treatment","contr.poly"))
fit<-rm.fit(I(1-y)~diagnosis+treatment+log(time,2),data=table.11.4,family="binary")
```

```
Coefficients:
                Value Std. Error t value
 (Intercept) -0.8856619 0.14775897  -5.994
   diagnosis  1.2880140 0.14681782   8.773
   treatment -0.8833318 0.14398722  -6.135
log(time, 2)  0.9013979 0.09007564  10.007

Log likelihood:  -596.2552
```

Here is the fit of the second model at the bottom of p. 397, which adds an occasion x treatment interaction. Again, I used 1-y as the response because of the way I coded an N and A above. I want to model the probability of a normal response.

```
fit<-rm.fit(I(1-y)~diagnosis+treatment*log(time,2),data=table.11.4,family="binary")
```

```
Coefficients:
                          Value Std. Error t value
           (Intercept) -1.40017270  0.1824999  -7.672
             diagnosis  1.31298541  0.1480636   8.868
             treatment  0.05900073  0.2231584   0.264
          log(time, 2)  1.49896932  0.1516299   9.886
treatment:log(time, 2) -1.01640792  0.1886942  -5.387

Log likelihood:  -580.8718

Message:  RELATIVE FUNCTION CONVERGENCE
```

`rm.fit` fits the model using maximum likelihood subject to constraints on the marginal distributions. Thus, the coefficients are not exactly the same.

Here are the fitted probabilities:

```
junk<-fit$fitted.values
probs<-junk[1:4,]      # take only the first four
dimnames(probs)<-list(trtcombn=c("mild-std","mild-new","sev-std","sev-new"),
   week=c("1","2","3"))
```

```
                1         2         3
mild-std 0.4929538 0.6116789 0.7184804
mild-new 0.4782170 0.8040469 0.9483754
 sev-std 0.2073174 0.2976297 0.4070801
 sev-new 0.1977887 0.5246791 0.8317059
```

Now, I illustrate the same analyses using library YAGS (Yet Another GEE Solver).

First, set up the data:

```
table.11.4<-cbind(rep(c("mild","severe"),c(16+13+9+3+14+4+15+6+31+0+6+0+22+2+9+0,
   2+2+8+9+9+15+27+28+7+2+5+2+31+5+32+6)),
   c(rep(c("standard","new"), c(16+13+9+3+14+4+15+6,31+0+6+0+22+2+9+0)),
   rep(c("standard","new"),c(2+2+8+9+9+15+27+28,7+2+5+2+31+5+32+6))))

resp<-matrix(c(0,0,0, 0,0,1, 0,1,0, 0,1,1, 1,0,0, 1,0,1, 1,1,0, 1,1,1),byrow=T,nc=3)
resp<-rbind(resp,resp,resp,resp)
n<-c(16,13,9,3,14,4,15,6,31,0,6,0,22,2,9,0,2,2,8,9,9,15,27,28,7,2,5,2,31,5,32,6)
resp<-cbind(resp,n)

temp<-matrix(unlist(apply(resp,1,function(x){
    if(x[4]!=0) t(matrix(rep(x[1:3],x[4]),nr=x[4],nc=3,byrow=T))
})),nc=3,byrow=T)

table.11.4<-data.frame(table.11.4,temp)
names(table.11.4)<-c("Diagnosis","Treatment","y1","y2","y3")
```

The first 100 values of the data frame are shown below. Again, y1, y2, and y3 are the three responses at the three occasions.

```
    Diagnosis Treatment y1 y2 y3
  1      mild  standard  0  0  0
  2      mild  standard  0  0  0
  3      mild  standard  0  0  0
  4      mild  standard  0  0  0
  5      mild  standard  0  0  0
  6      mild  standard  0  0  0
  7      mild  standard  0  0  0
  8      mild  standard  0  0  0
  9      mild  standard  0  0  0
 10      mild  standard  0  0  0
 11      mild  standard  0  0  0
 12      mild  standard  0  0  0
 13      mild  standard  0  0  0
 14      mild  standard  0  0  0
 15      mild  standard  0  0  0
 16      mild  standard  0  0  0
 17      mild  standard  0  0  1
 18      mild  standard  0  0  1
 19      mild  standard  0  0  1
 20      mild  standard  0  0  1
 21      mild  standard  0  0  1
 22      mild  standard  0  0  1
 23      mild  standard  0  0  1
 24      mild  standard  0  0  1
 25      mild  standard  0  0  1
 26      mild  standard  0  0  1
 27      mild  standard  0  0  1
 28      mild  standard  0  0  1
 29      mild  standard  0  0  1
 30      mild  standard  0  1  0
 31      mild  standard  0  1  0
 32      mild  standard  0  1  0
 33      mild  standard  0  1  0
```

34	mild	standard	0	1	0
35	mild	standard	0	1	0
36	mild	standard	0	1	0
37	mild	standard	0	1	0
38	mild	standard	0	1	0
39	mild	standard	0	1	1
40	mild	standard	0	1	1
41	mild	standard	0	1	1
42	mild	standard	1	0	0
43	mild	standard	1	0	0
44	mild	standard	1	0	0
45	mild	standard	1	0	0
46	mild	standard	1	0	0
47	mild	standard	1	0	0
48	mild	standard	1	0	0
49	mild	standard	1	0	0
50	mild	standard	1	0	0
51	mild	standard	1	0	0
52	mild	standard	1	0	0
53	mild	standard	1	0	0
54	mild	standard	1	0	0
55	mild	standard	1	0	0
56	mild	standard	1	0	1
57	mild	standard	1	0	1
58	mild	standard	1	0	1
59	mild	standard	1	0	1
60	mild	standard	1	1	0
61	mild	standard	1	1	0
62	mild	standard	1	1	0
63	mild	standard	1	1	0
64	mild	standard	1	1	0
65	mild	standard	1	1	0
66	mild	standard	1	1	0
67	mild	standard	1	1	0
68	mild	standard	1	1	0
69	mild	standard	1	1	0
70	mild	standard	1	1	0
71	mild	standard	1	1	0
72	mild	standard	1	1	0
73	mild	standard	1	1	0
74	mild	standard	1	1	0
75	mild	standard	1	1	1
76	mild	standard	1	1	1
77	mild	standard	1	1	1
78	mild	standard	1	1	1
79	mild	standard	1	1	1
80	mild	standard	1	1	1
81	mild	new	0	0	0
82	mild	new	0	0	0
83	mild	new	0	0	0
84	mild	new	0	0	0
85	mild	new	0	0	0
86	mild	new	0	0	0
87	mild	new	0	0	0
88	mild	new	0	0	0
89	mild	new	0	0	0
90	mild	new	0	0	0
91	mild	new	0	0	0
92	mild	new	0	0	0
93	mild	new	0	0	0
94	mild	new	0	0	0
95	mild	new	0	0	0
96	mild	new	0	0	0
97	mild	new	0	0	0
98	mild	new	0	0	0
99	mild	new	0	0	0
100	mild	new	0	0	0

Now, I have to add the subject “id” and “week” factors. “id” indicates the subject tested. The “week” number is either 1,2, or 4.

```
temp<-table.11.4[rep(1:nrow(table.11.4),rep(3,nrow(table.11.4))),1:2]
row.names(temp)<-1:nrow(temp)
table.11.4.yags<-cbind.data.frame(temp,c(t(data.matrix(table.11.4)[,3:5])))
```

add id factor:

```
table.11.4.yags<-cbind.data.frame(rep(1:nrow(table.11.4),rep(3,nrow(table.11.4))),
   table.11.4.yags)
```

add time (in weeks) factor:

```
table.11.4.yags<-cbind.data.frame(table.11.4.yags,rep(c(1,2,4),nrow(table.11.4)))
names(table.11.4.yags)[c(1,4,5)]<-c("id","suffering","week")
```

Here are the first 100 values:

	id	Diagnosis	Treatment	suffering	week
1	1	mild	standard	0	1
2	1	mild	standard	0	2
3	1	mild	standard	0	4
4	2	mild	standard	0	1
5	2	mild	standard	0	2
6	2	mild	standard	0	4
7	3	mild	standard	0	1
8	3	mild	standard	0	2
9	3	mild	standard	0	4
10	4	mild	standard	0	1
11	4	mild	standard	0	2
12	4	mild	standard	0	4
13	5	mild	standard	0	1
14	5	mild	standard	0	2
15	5	mild	standard	0	4
16	6	mild	standard	0	1
17	6	mild	standard	0	2
18	6	mild	standard	0	4
19	7	mild	standard	0	1
20	7	mild	standard	0	2
21	7	mild	standard	0	4
22	8	mild	standard	0	1
23	8	mild	standard	0	2
24	8	mild	standard	0	4
25	9	mild	standard	0	1
26	9	mild	standard	0	2
27	9	mild	standard	0	4
28	10	mild	standard	0	1
29	10	mild	standard	0	2
30	10	mild	standard	0	4
31	11	mild	standard	0	1
32	11	mild	standard	0	2
33	11	mild	standard	0	4
34	12	mild	standard	0	1
35	12	mild	standard	0	2
36	12	mild	standard	0	4
37	13	mild	standard	0	1
38	13	mild	standard	0	2
39	13	mild	standard	0	4
40	14	mild	standard	0	1
41	14	mild	standard	0	2
42	14	mild	standard	0	4
43	15	mild	standard	0	1

```
  44  15      mild  standard         0    2
  45  15      mild  standard         0    4
  46  16      mild  standard         0    1
  47  16      mild  standard         0    2
  48  16      mild  standard         0    4
  49  17      mild  standard         0    1
  50  17      mild  standard         0    2
  51  17      mild  standard         1    4
  52  18      mild  standard         0    1
  53  18      mild  standard         0    2
  54  18      mild  standard         1    4
  55  19      mild  standard         0    1
  56  19      mild  standard         0    2
  57  19      mild  standard         1    4
  58  20      mild  standard         0    1
  59  20      mild  standard         0    2
  60  20      mild  standard         1    4
  61  21      mild  standard         0    1
  62  21      mild  standard         0    2
  63  21      mild  standard         1    4
  64  22      mild  standard         0    1
  65  22      mild  standard         0    2
  66  22      mild  standard         1    4
  67  23      mild  standard         0    1
  68  23      mild  standard         0    2
  69  23      mild  standard         1    4
  70  24      mild  standard         0    1
  71  24      mild  standard         0    2
  72  24      mild  standard         1    4
  73  25      mild  standard         0    1
  74  25      mild  standard         0    2
  75  25      mild  standard         1    4
  76  26      mild  standard         0    1
  77  26      mild  standard         0    2
  78  26      mild  standard         1    4
  79  27      mild  standard         0    1
  80  27      mild  standard         0    2
  81  27      mild  standard         1    4
  82  28      mild  standard         0    1
  83  28      mild  standard         0    2
  84  28      mild  standard         1    4
  85  29      mild  standard         0    1
  86  29      mild  standard         0    2
  87  29      mild  standard         1    4
  88  30      mild  standard         0    1
  89  30      mild  standard         1    2
  90  30      mild  standard         0    4
  91  31      mild  standard         0    1
  92  31      mild  standard         1    2
  93  31      mild  standard         0    4
  94  32      mild  standard         0    1
  95  32      mild  standard         1    2
  96  32      mild  standard         0    4
  97  33      mild  standard         0    1
  98  33      mild  standard         1    2
  99  33      mild  standard         0    4
 100  34      mild  standard         0    1
```

Now, fit the second model:

```
attach(table.11.4.yags)
time<-log(week,2)

library(yags)
```

```
fit<-yags(I(1-suffering)~factor(Diagnosis,levels=c("severe","mild"))
   +factor(Treatment,levels=c("new","standard"))*time,id=id,cor.met=week,corstr="uns
   tructured",family=binomial)

summary(fit)

Coefficients:
                 Estimate Naive S.E.    Naive z Robust S.E.    Robust z
   (Intercept) -1.38484307  0.1836080 -7.5423910   0.1772216  -7.8141883
     Diagnosis  1.30489234  0.1448783  9.0068171   0.1450193   8.9980583
     Treatment  0.05441048  0.2261665  0.2405771   0.2271399   0.2395461
          time  1.48890188  0.1493473  9.9693953   0.1440043  10.3392898
Treatment:time -1.01299682  0.1870708 -5.4150437   0.1865475  -5.4302367

Estimated Scale Parameter:  0.982367
Number of Iterations:  2
```

And, I obtain the fitted values as follows:

```
(fitted(fit))[1:3,]
[1] 0.2002323 0.5259913 0.8310322

(fitted(fit))[340:342,]
[1] 0.2090878 0.2984840 0.4064592

(fitted(fit))[679:681,]
[1] 0.4800230 0.8036005 0.9477402

fitted(fit)[1018:1020,]
[1] 0.4936153 0.6107260 0.7163179
```

D. Modeling a Repeated Ordinal Response

Agresti uses the Insomnia data on p. 402 (Table 11.7) to illustrate modeling a repeated ordinal response. As with the ordered mean response models in Chapter 9, I could not find a package that could implement both the ordinal and repeated nature of the response. Thus, I compute the weighted least squares estimates of the coefficients in (11.11) directly using the development on pages 458-462 of Agresti.

First, I set up the data and sample proportion vectors for each Occasion x Treatment combination.

```
table.11.7<-
   matrix(c(7,4,1,0,11,5,2,2,13,23,3,1,9,17,13,8,7,4,2,1,14,5,1,0,6,9,18,2,4,11,14,2
   2),nc=4,nr=8,byrow=T)
p1<-(rowSums(table.11.7[1:4,1:4])/sum(table.11.7[1:4,1:4]))
p2<-(colSums(table.11.7[1:4,1:4])/sum(table.11.7[1:4,1:4]))
p3<-(rowSums(table.11.7[5:8,1:4])/sum(table.11.7[5:8,1:4]))
p4<-(colSums(table.11.7[5:8,1:4])/sum(table.11.7[5:8,1:4]))

n1<-119; n2<-n1; n3<-120; n4<-n3 # sample sizes for each "population"
n<-c(n1,n2,n3,n4)
J<-4   # We have four response variables
I<-4   # We have four partly dependent multinomial samples
```

The 4 x 1 vector $\mathbf{p}_i$ represents the sample distribution of the response for the ith population. The populations are ordered 1 through 4 according to their presentation in Table 11.8 on p. 403. Thus, group Active/Initial is group 1, and Passive/Follow-up is group 4.

The variance-covariance matrix on p. 459 in Agresti is modified here to account for the fact that the first and second populations and the third and fourth populations are dependent (they are the same individuals). Thus, the covariance matrix becomes

$$\mathbf{V} = \begin{pmatrix} \mathbf{V}_1 & \mathbf{V}_{12} & \mathbf{0} & \mathbf{0} \\ \mathbf{V}_{12} & \mathbf{V}_2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{V}_3 & \mathbf{V}_{34} \\ \mathbf{0} & \mathbf{0} & \mathbf{V}_{34} & \mathbf{V}_4 \end{pmatrix}$$

where $\mathbf{V}_i$ is as described on p. 459 in Agresti, and

$$n_i \mathbf{V}_{ii'} = \begin{pmatrix} \text{cov}(y_{1|i}, y_{1|i'}) & \text{cov}(y_{1|i}, y_{2|i'}) & \text{cov}(y_{1|i}, y_{3|i'}) & \text{cov}(y_{1|i}, y_{4|i'}) \\ \text{cov}(y_{2|i}, y_{1|i'}) & \text{cov}(y_{2|i}, y_{2|i'}) & \text{cov}(y_{2|i}, y_{3|i'}) & \text{cov}(y_{2|i}, y_{4|i'}) \\ \text{cov}(y_{3|i}, y_{1|i'}) & \text{cov}(y_{3|i}, y_{2|i'}) & \text{cov}(y_{3|i}, y_{3|i'}) & \text{cov}(y_{3|i}, y_{4|i'}) \\ \text{cov}(y_{4|i}, y_{1|i'}) & \text{cov}(y_{4|i}, y_{2|i'}) & \text{cov}(y_{4|i}, y_{3|i'}) & \text{cov}(y_{4|i}, y_{4|i'}) \end{pmatrix}$$

where $i' \neq i$. The covariance terms in the above matrix are

$$\text{cov}(y_{k|i}, y_{l|i'}) = E(y_{k|i} y_{l|i'}) - E(y_{k|i})\,E(y_{l|i'}) = \Pr(y_{k|i} = 1,\ y_{l|i'} = 1) - \Pr(y_{k|i} = 1)\Pr(y_{l|i'} = 1)$$

These terms can be estimated using the sample proportions.

Thus, the Splus commands to input the covariance matrix **V** are

```
Vnames<-numeric(length=I)
pnames<-Vnames

for(i in 1:I)
{
   Vnames[i] <- paste("V", i, collapse = " ", sep = "")
   pnames[i]<-paste("p",i,collapse=" ",sep="")
   V<-matrix(0,nc=J,nr=J)
   p<-eval(parse(text = pnames[i]))
   diag(V)<-p*(1-p)
   p<-as.matrix(p)
   junk<-matrix(-kronecker(p,p),nc=J,nr=J,byrow=T)
   V[lower.tri(diag(J))]<-junk[lower.tri(junk)]
   V<-t(V)
   V[lower.tri(V)]<-junk[lower.tri(junk,diag=F)]
   assign(Vnames[i], matrix(V/n[i], ncol = J, byrow = T))
}

# Compute V12
active<-table.11.7[1:4,1:4]
active<-sweep(active,1,rowSums(active),FUN="/")
p<-cbind(p1,p2,p3,p4)

V12<-matrix(c(p1[1]*(active[1,]-p[,2]),p1[2]*(active[2,]-p[,2]),p1[3]*(active[3,]-
   p[,2]),p1[4]*(active[4,]-p[,2])), nc=J,nr=J,byrow=T)
V12<-V12/n1

# Compute V34
passive<-table.11.7[5:8,1:4]
```

```
passive<-sweep(passive,1,rowSums(passive),FUN="/")
V34<-matrix(c(p3[1]*(passive[1,]-p[,4]),p3[2]*(passive[2,]-
   p[,4]),p3[3]*(passive[3,]-p[,4]),p3[4]*(passive[4,]-p[,4])), nc=J,nr=J,byrow=T)
V34<-V34/n3

# Compute V
zero<-matrix(0,J,J)
V<-rbind(
cbind(V1,V12,zero,zero),
cbind(t(V12),V2,zero,zero),
cbind(zero,zero,V3,V34),
cbind(zero,zero,t(V34),V4))
```

The objects Vnames and pnames have the same usage as in Section 9.d.

The model (equation 13.11) for this example encompasses the 12 equations

$$F_1(\boldsymbol{\pi}) = a_1 - b_1^O - b_1^T - g_{11} = \log\left(\frac{p_{1|1}}{p_{2|1}+p_{3|1}+p_{4|1}}\right)$$

$$F_2(\boldsymbol{\pi}) = a_2 - b_1^O - b_1^T - g_{11} = \log\left(\frac{p_{1|1}+p_{2|1}}{p_{3|1}+p_{4|1}}\right)$$

$$\vdots$$

$$F_{11}(\boldsymbol{\pi}) = a_2 = \log\left(\frac{p_{1|4}+p_{2|4}}{p_{3|4}+p_{4|4}}\right)$$

$$F_{12}(\boldsymbol{\pi}) = a_3 = \log\left(\frac{p_{1|4}+p_{2|4}+p_{3|4}}{p_{4|4}}\right),$$

one for each of the four treatment combinations and three response cutoffs. The matrix **Q** of first derivatives of the response functions *F*, has the following form

$$\mathbf{Q} = \begin{pmatrix} \frac{\partial}{\partial p_{1|1}} F_1(\boldsymbol{\pi}) & \cdots & \frac{\partial}{\partial p_{4|4}} F_1(\boldsymbol{\pi}) \\ \vdots & & \vdots \\ \frac{\partial}{\partial p_{1|1}} F_{12}(\boldsymbol{\pi}) & \cdots & \frac{\partial}{\partial p_{4|4}} F_{12}(\boldsymbol{\pi}) \end{pmatrix}$$

and is straightforward to compute using the function `deriv` or `deriv3` available from the MASS library. The response functions and the matrix of derivatives can be estimated using sample values. Here are the Splus commands to compute them for this example.

```
library(MASS)

# Definitions of the derivative functions in the Q matrix:
d1F<-deriv3(~log(x/(a+b+k)),c("x"),function(x,a,b,k){},hessian=F)
d2F<-deriv3(~log(a/(x+b+k)),c("x"),function(x,a,b,k){},hessian=F)

d3F<-deriv3(~log((x+a)/(b+k)),c("x"),function(x,a,b,k){},hessian=F)
d4F<-deriv3(~log((b+a)/(x+k)),c("x"),function(x,a,b,k){},hessian=F)

d5F<-deriv3(~log((x+a+b)/k),c("x"),function(x,a,b,k){},hessian=F)
d6F<-deriv3(~log((k+a+b)/x),c("x"),function(x,a,b,k){},hessian=F)

# Now, we plug in the specific estimated probabilities:
Q<-c(
as.numeric(attr(d1F(p1[1],p1[2],p1[3],p1[4]),"gradient")),
as.numeric(attr(d2F(p1[2],p1[1],p1[3],p1[4]),"gradient")),
```

```
as.numeric(attr(d2F(p1[3],p1[1],p1[2],p1[4]),"gradient")),
as.numeric(attr(d2F(p1[4],p1[1],p1[2],p1[3]),"gradient")),
rep(0,12),

as.numeric(attr(d3F(p1[1],p1[2],p1[3],p1[4]),"gradient")),
as.numeric(attr(d3F(p1[2],p1[1],p1[3],p1[4]),"gradient")),
as.numeric(attr(d4F(p1[3],p1[1],p1[2],p1[4]),"gradient")),
as.numeric(attr(d4F(p1[4],p1[1],p1[2],p1[3]),"gradient")),
rep(0,12),

as.numeric(attr(d5F(p1[1],p1[2],p1[3],p1[4]),"gradient")),
as.numeric(attr(d5F(p1[2],p1[1],p1[3],p1[4]),"gradient")),
as.numeric(attr(d5F(p1[3],p1[1],p1[2],p1[4]),"gradient")),
as.numeric(attr(d6F(p1[4],p1[1],p1[2],p1[3]),"gradient")),
rep(0,12),

rep(0,4),
as.numeric(attr(d1F(p2[1],p2[2],p2[3],p2[4]),"gradient")),
as.numeric(attr(d2F(p2[2],p2[1],p2[3],p2[4]),"gradient")),
as.numeric(attr(d2F(p2[3],p2[1],p2[2],p2[4]),"gradient")),
as.numeric(attr(d2F(p2[4],p2[1],p2[2],p2[3]),"gradient")),
rep(0,8),

rep(0,4),
as.numeric(attr(d3F(p2[1],p2[2],p2[3],p2[4]),"gradient")),
as.numeric(attr(d3F(p2[2],p2[1],p2[3],p2[4]),"gradient")),
as.numeric(attr(d4F(p2[3],p2[1],p2[2],p2[4]),"gradient")),
as.numeric(attr(d4F(p2[4],p2[1],p2[2],p2[3]),"gradient")),
rep(0,8),

rep(0,4),
as.numeric(attr(d5F(p2[1],p2[2],p2[3],p2[4]),"gradient")),
as.numeric(attr(d5F(p2[2],p2[1],p2[3],p2[4]),"gradient")),
as.numeric(attr(d5F(p2[3],p2[1],p2[2],p2[4]),"gradient")),
as.numeric(attr(d6F(p2[4],p2[1],p2[2],p2[3]),"gradient")),
rep(0,8),

rep(0,8),
as.numeric(attr(d1F(p3[1],p3[2],p3[3],p3[4]),"gradient")),
as.numeric(attr(d2F(p3[2],p3[1],p3[3],p3[4]),"gradient")),
as.numeric(attr(d2F(p3[3],p3[1],p3[2],p3[4]),"gradient")),
as.numeric(attr(d2F(p3[4],p3[1],p3[2],p3[3]),"gradient")),
rep(0,4),

rep(0,8),
as.numeric(attr(d3F(p3[1],p3[2],p3[3],p3[4]),"gradient")),
as.numeric(attr(d3F(p3[2],p3[1],p3[3],p3[4]),"gradient")),
as.numeric(attr(d4F(p3[3],p3[1],p3[2],p3[4]),"gradient")),
as.numeric(attr(d4F(p3[4],p3[1],p3[2],p3[3]),"gradient")),
rep(0,4),

rep(0,8),
as.numeric(attr(d5F(p3[1],p3[2],p3[3],p3[4]),"gradient")),
as.numeric(attr(d5F(p3[2],p3[1],p3[3],p3[4]),"gradient")),
as.numeric(attr(d5F(p3[3],p3[1],p3[2],p3[4]),"gradient")),
as.numeric(attr(d6F(p3[4],p3[1],p3[2],p3[3]),"gradient")),
rep(0,4),

rep(0,12),
as.numeric(attr(d1F(p4[1],p4[2],p4[3],p4[4]),"gradient")),
as.numeric(attr(d2F(p4[2],p4[1],p4[3],p4[4]),"gradient")),
as.numeric(attr(d2F(p4[3],p4[1],p4[2],p4[4]),"gradient")),
```

```
as.numeric(attr(d2F(p4[4],p4[1],p4[2],p4[3]),"gradient")),

rep(0,12),
as.numeric(attr(d3F(p4[1],p4[2],p4[3],p4[4]),"gradient")),
as.numeric(attr(d3F(p4[2],p4[1],p4[3],p4[4]),"gradient")),
as.numeric(attr(d4F(p4[3],p4[1],p4[2],p4[4]),"gradient")),
as.numeric(attr(d4F(p4[4],p4[1],p4[2],p4[3]),"gradient")),

rep(0,12),
as.numeric(attr(d5F(p4[1],p4[2],p4[3],p4[4]),"gradient")),
as.numeric(attr(d5F(p4[2],p4[1],p4[3],p4[4]),"gradient")),
as.numeric(attr(d5F(p4[3],p4[1],p4[2],p4[4]),"gradient")),
as.numeric(attr(d6F(p4[4],p4[1],p4[2],p4[3]),"gradient"))
)

Q<-matrix(Q,nr=12,nc=16,byrow=T)

# Definitions of the response functions, afterward evaluated at the sample proportions:
f1<-function(x,a,b,k)log(x/(a+b+k))
f2<-function(x,a,b,k)log((x+a)/(b+k))
f3<-function(x,a,b,k)log((x+a+b)/k)

Fp<-c( f1(p1[1],p1[2],p1[3],p1[4]), f2(p1[1],p1[2],p1[3],p1[4]),
       f3(p1[1],p1[2],p1[3],p1[4]), f1(p2[1],p2[2],p2[2],p2[4]),
       f2(p2[1],p2[2],p2[3],p2[4]), f3(p2[1],p2[2],p2[3],p2[4]),
       f1(p3[1],p3[2],p3[3],p3[4]), f2(p3[1],p3[2],p3[3],p3[4]),
       f3(p3[1],p3[2],p3[3],p3[4]), f1(p4[1],p4[2],p4[3],p4[4]),
       f2(p4[1],p4[2],p4[3],p4[4]), f3(p4[1],p4[2],p4[3],p4[4]) )
```

The transformed covariance matrix is estimated as before:

```
VF<-Q%*%V%*%t(Q)
```

The design matrix reflects the model equation, $F(\boldsymbol{\pi}) = \mathbf{X}\boldsymbol{\beta}$, with $\boldsymbol{\beta} = (a_1,\ a_2,\ a_3,\ b_1^O,\ b_1^T,\ g_{11})^{\mathrm{T}}$,

```
X<-rbind(diag(3),diag(3),diag(3),diag(3))
X<-cbind(X,rep(rep(c(-1,0),c(3,3)),2),rep(c(-1,0),c(6,6)),rep(c(-1,0),c(3,9)))
```

```
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    1    0    0   -1   -1   -1
 [2,]    0    1    0   -1   -1   -1
 [3,]    0    0    1   -1   -1   -1
 [4,]    1    0    0    0   -1    0
 [5,]    0    1    0    0   -1    0
 [6,]    0    0    1    0   -1    0
 [7,]    1    0    0   -1    0    0
 [8,]    0    1    0   -1    0    0
 [9,]    0    0    1   -1    0    0
[10,]    1    0    0    0    0    0
[11,]    0    1    0    0    0    0
[12,]    0    0    1    0    0    0
```

To estimate beta, we use the weighted least squares formula:

```
InvVF<-solve(VF)
Covb<-solve(t(X)%*%InvVF%*%X)
b<-InvVF%*%t(X)%*%solve(VF)%*%Fp
```

```
 [1] -1.16085340  0.09803907  1.37253480  1.05430567 -0.69429003  0.65432148
```

The residual chi-squared statistic is then computed as

```
t(Fp-X%*%b)%*%InvVF%*%(Fp-X%*%b)
```

```
[1] 7.417129
```

The ASEs for all estimates are

```
sqrt(diag(Covb))
[1] 0.1728720 0.1705049 0.1829844 0.1604752 0.2345762 0.2460542
```

Obtaining predicted values requires just a little more thought than usual. Because we are modeling cumulative logits, the probabilities estimated by

```
cprob<-exp(X%*%b)/(1+exp(X%*%b))
```

are the cumulative probabilities: $\Pr(y \le j \mid \mathbf{x})$. So, to get the probabilities $\Pr(y = j \mid \mathbf{x})$, we just take successive differences of cumulative probabilities where appropriate.

```
prob<-matrix(c(cprob[1],diff(cprob[1:3]),1-cprob[3],cprob[4], diff(cprob[4:6]), 1-
   cprob[6]),byrow=T,nc=4)
prob2<-matrix(c(cprob[7],diff(cprob[7:9]),1-cprob[9],cprob[10], diff(cprob[10:12]),
   1-cprob[12]),byrow=T,nc=4)
prob<-array(c(prob,prob2),dim=c(2,4,2))
dimnames(prob)<-list(c("Initial","Follow-up"),c("<20","20-30","30-60",">60"),
   c("Active","Passive"))
```

```
, , Active
                <20     20-30     30-60       >60
  Initial 0.1020006 0.1837122 0.3028913 0.4113959
Follow-up 0.3854300 0.3029012 0.1993055 0.1123633

, , Passive
                 <20     20-30     30-60       >60
  Initial 0.09839743 0.1792289 0.3012663 0.4211074
Follow-up 0.23851225 0.2859779 0.2732992 0.2022106
```

E. Markov Chain Models

Agresti fits several Markov chain models to the respiratory illness data in Table 11.9. These can be fit as ordinary log-linear models with a function like `glm`.

```
table.11.9<-expand.grid(twelve=(c(1,2)),eleven=(c(1,2)),ten=(c(1,2)), nine=(c(1,2)))
table.11.9<-cbind.data.frame(table.11.9,
   counts=c(94,30,15,28,14,9,12,63,19,15,10,44, 17,42,35,572))
```

```
   twelve eleven ten nine counts
 1      1      1   1    1     94
 2      2      1   1    1     30
 3      1      2   1    1     15
 4      2      2   1    1     28
 5      1      1   2    1     14
 6      2      1   2    1      9
 7      1      2   2    1     12
 8      2      2   2    1     63
 9      1      1   1    2     19
10      2      1   1    2     15
11      1      2   1    2     10
```

```
12      2      2   1    2     44
13      1      1   2    2     17
14      2      1   2    2     42
15      1      2   2    2     35
16      2      2   2    2    572
```

Test a first-order model:

```
glm(counts~nine*ten+ten*eleven+eleven*twelve,data=table.11.9,family=poisson(log))

Degrees of Freedom: 16 Total; 8 Residual
Residual Deviance: 122.9025
```

Test a second-order model:

```
glm(counts~nine*ten*eleven+ten*eleven*twelve,data=table.11.9,family=poisson(log))

Degrees of Freedom: 16 Total; 4 Residual
Residual Deviance: 23.86324
```

Test a model that permits all pairwise associations:

```
options(contrasts=c("contr.sum","contr.poly"))
glm(counts~nine*ten+nine*eleven+nine*twelve+ten*eleven+ten*twelve+eleven*twelve,data
   =table.11.9,family=poisson(log))

Coefficients:
 (Intercept)      nine       ten    eleven    twelve nine:ten nine:eleven
      nine:twelve
    19.50572 -5.391511 -6.488897 -6.244144 -5.207749 1.806361   0.9478092
   1.053104

 ten:eleven ten:twelve eleven:twelve
   1.645769   1.074206      1.849747

Degrees of Freedom: 16 Total; 5 Residual
Residual Deviance: 1.458497
```

Test a model that restricts pairwise associations to be one of two values (high or low) depending on whether the association is first- or second-order:

I first change the coding in table.11.9 to be 0/1 instead of 1/2:

```
table.11.9[,1:4]<-table.11.9[,1:4]-1
attach(table.11.9)
```

The objects pairs and thirdpairs represent the common effects of first- and second-order associations.

```
pairs<-cbind((nine*ten),(ten*eleven),(eleven*twelve))
thirdpairs<-cbind((ten*twelve),nine*eleven,nine*twelve)
pairs<-as.numeric(rowSums(pairs))
thirdpairs<-as.numeric(rowSums(thirdpairs))

glm(counts~nine + ten + eleven + twelve +  I(thirdpairs) + I(pairs),
      data=table.11.9,family=poisson(log))

Coefficients:
 (Intercept)      nine       ten    eleven   twelve I(thirdpairs) I(pairs)
    4.546055 -1.596589 -1.970608 -1.853073 -1.13977      1.035713  1.75203
```

```
Degrees of Freedom: 16 Total; 9 Residual
Residual Deviance: 2.268529
```

The estimated log odds ratios are highlighted.