目录

[1. Poisson Regression(6.6) 2](#_Toc28943393)

[1.1 a——rwmetrop and analysis 3](#_Toc28943394)

[1.2 b——indepmetrop 5](#_Toc28943395)

[1.3 c——Gibbs 6](#_Toc28943396)

[1.4 Comparison 7](#_Toc28943397)

[1.5 MAP 8](#_Toc28943398)

[2. Inference about the Box-Cox transformation(6.10) 10](#_Toc28943399)

[2.1 a 11](#_Toc28943400)

[2.2 b 11](#_Toc28943401)

[2.3 c 11](#_Toc28943402)

[2.4 d 12](#_Toc28943403)

[2.5 e 12](#_Toc28943404)

[3. Is a basketball player streaky? (8.4) 13](#_Toc28943405)

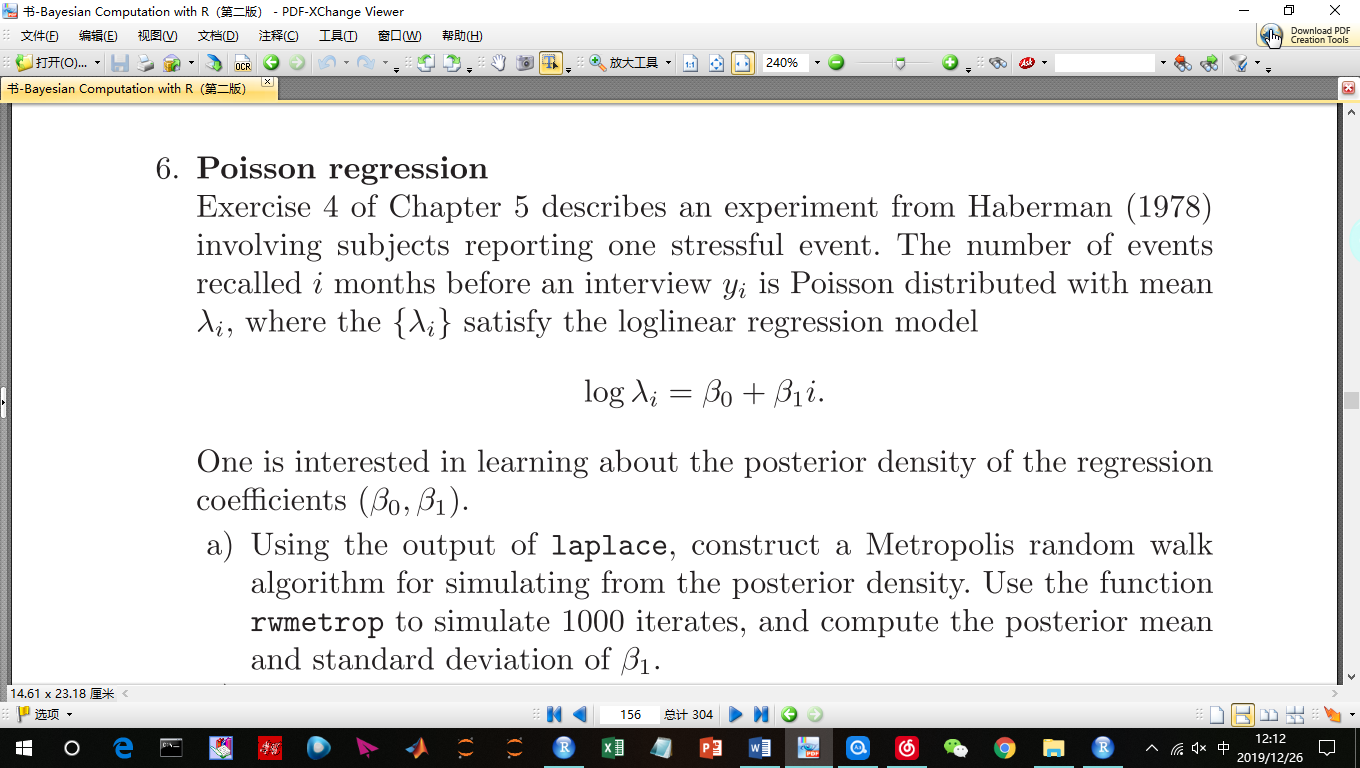
[4. Test of independence (example from Agresti and Franklin (2005)) (8.5) 15](#_Toc28943406)

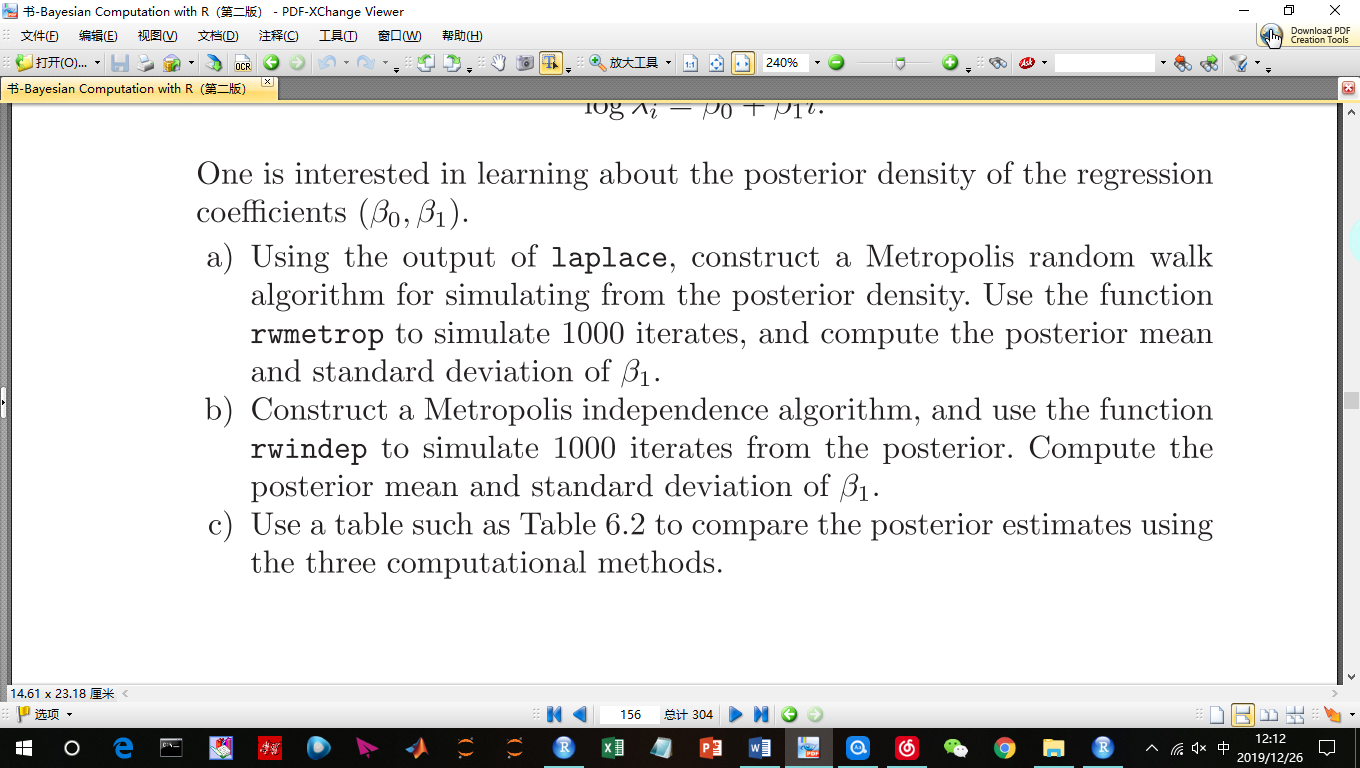
[4.1 a 15](#_Toc28943407)

[4.2 b 16](#_Toc28943408)

[4.3 c 16](#_Toc28943409)

# Poisson Regression(6.6)





在这里，我们的数据服从均值为的poisson分布，即：

其中：

即是：

所以，当参数未知时，其条件概率为：

似然函数（）为：

Log似然为：

由于固定，其似然函数正比于：

由题意，给参数施加一个均匀先验后，其最终的后验概率为：

## 1.1 a——rwmetrop and analysis

在这里，我们仿照本章6.7节给出解答过程。

首先，写出所求解出的后验概率函数，随后规定模拟次数与burn-in period。由于在实验中发现模拟次数sim=1000时结果的不稳定性很大，为方便比较，我们模拟10000次以获得更稳定的效果。

rm(list=ls())

library(LearnBayes)

data=c(15,11,14,17,5,11,10,4,8,10,7,9,11,3,6,1,1,4)

f=function(beta,data){

beta0=beta[1]

beta1=beta[2]

i=1:18

y=data

sum((beta0+beta1\*i)\*y-exp(beta0+beta1\*i))

}

sim=10000

burnin=2000

题目首先让我们使用laplace函数。我们采取一个小技巧（类似6.7节找初值的方法）来获取一个较好的初值。我们知道服从均值为的poisson分布，由于：

我们采用替代：

通过带入()的两组数据(1,15),(9,8)求得一组()=( 127/8, -7/8)，我们使用这组参数值作为初值。求得laplace的mode与var。

start2=c(127/8,-7/8)

fit=laplace(f,start2,data)

fit

$mode

[1] 2.79392439 -0.08230398

$var

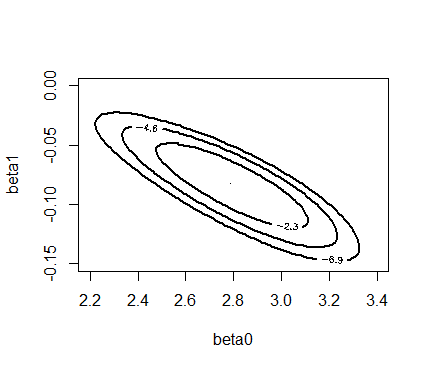
[,1] [,2]

[1,] 0.02200785 -0.002066740

[2,] -0.00206674 0.000280728

接下来，通过Laplace输出的mode我们利用mycontour函数多次调试后画出区域图：

mycontour(f,c(2.2,3.4,-0.15,0),data,xlab="beta0",ylab="beta1")



接下来，我们通过laplace函数的输出设置随机游走模拟的proposal参数，进行10000次模拟，并计算接受率。此接受率在25%-45%之间，我们比较满意。

proposal=list(var=fit$var,scale=2)

fit2=rwmetrop(f,proposal,start1,sim,data)

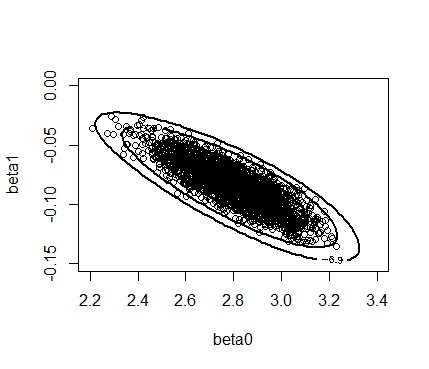
fit2$accept

[1] 0.2873

随后，我们将模拟的点除去burn-in period后（如果不去除burn-in preiod会有一些离群点），在图上展示：

mycontour(f,c(2.2,3.4,-0.15,0),data,xlab="beta0",ylab="beta1")

points(fit2$par[burnin:sim, 1], fit2$par[burnin:sim, 2])



接下来，我们仿照6.7节给出随机游走模拟与laplace近似的对比，可以看出，无论是的均值与方差，两种方法差别都很小。

post.means=apply(fit2$par,2,mean)

post.sds=apply(fit2$par,2,sd)

modal.sds=sqrt(diag(fit$var))

> cbind(c(fit$mode), modal.sds)

modal.sds

[1,] 2.79392439 0.14835042

[2,] -0.08230398 0.01675494

> cbind(post.means, post.sds)

post.means post.sds

[1,] 2.80483591 0.17222863

[2,] -0.08355152 0.01775821

接下来，我们仿照6.8节给出MCMC模拟的输出分析：（依旧丢弃了burn-in period），从模拟的轨迹图和自相关图我们都能发现模拟的效果很好。

library(coda)

library(lattice)

dimnames(fit2$par)[[2]]=c("beta0", "beta1")

xyplot(mcmc(fit2$par[-c(1:burnin), ]), col = "black")

par(mfrow = c(2, 1))

autocorr.plot(mcmc(fit2$par[-c(1:burnin), ]), auto.layout = FALSE)

|  |  |
| --- | --- |
|  |  |

最后，我们输出题目要求的的后验均值和方差

> mean(fit2$par[burnin:sim,2])

[1] -0.08398493

> sd(fit2$par[burnin:sim,2])

[1] 0.01683423

## 1.2 b——indepmetrop

接下来，我们使用MCMC算法的独立模型，首先依旧从laplace的输出中获得proposal的参数，依旧进行10000次模拟，burn-in period为2000。接受率为0.9118。之后画出模拟图。

proposal2=list(var = fit$var, mu = t(fit$mode))

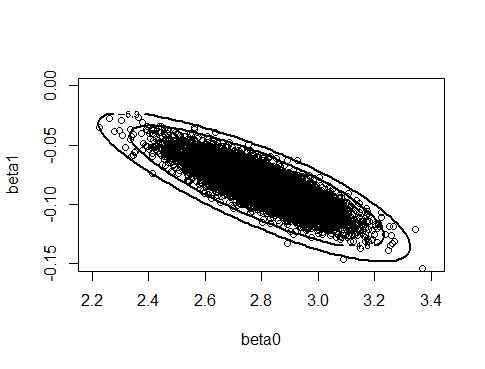
fit3=indepmetrop(f, proposal2, start2, sim, data)

fit3$accept

[1,] 0.9118

mycontour(f,c(2.2,3.4,-0.15,0),data,xlab="beta0",ylab="beta1")

points(fit3$par[burnin:sim, 1], fit3$par[burnin:sim, 2])



最后，我们输出题目要求的的后验均值和方差

> mean(fit3$par[burnin:sim,2])

[1] -0.08432691

> sd(fit3$par[burnin:sim,2])

[1] 0.016868

## 1.3 c——Gibbs

接下来，我们利用gibbs采样，通过调整参数得到了一组可以接受的接受率。之后画出模拟图。

> fit4=gibbs(f, start2, sim, c(0.3, 0.05), data)

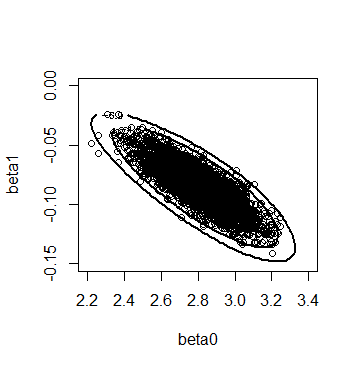
> fit4$accept

[,1] [,2]

[1,] 0.3161 0.2386

mycontour(f,c(2.2,3.4,-0.15,0),data,xlab="beta0",ylab="beta1")

points(fit4$par[burnin:sim, 1], fit4$par[burnin:sim, 2])



Gibbs采样的的后验均值和方差

> mean(fit4$par[burnin:sim,2])

[1] -0.08436472

> sd(fit4$par[burnin:sim,2])

[1] 0.01737734

## 1.4 Comparison

我们画出的四种方法模拟出的后验密度图，可以看到除了随机游走方法，其他方法都非常近似于正态分布。

plot(density(rnorm(sim,fit$mode[2],sqrt(fit$var[2,2]))),

type="l",col=4,lwd=2,lty=1,main="Simulation Comparison",

xlab="beta1",ylab="posterior density",ylim=c(0,30))

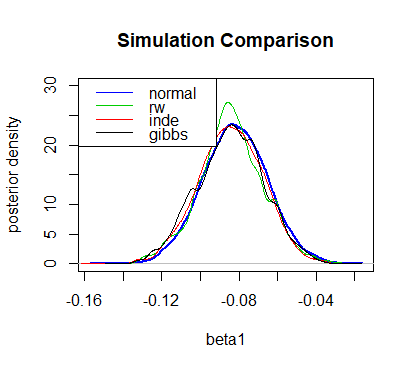
lines(density(fit2$par[burnin:sim,2]),type="l",col=3,lwd=1)

lines(density(fit3$par[burnin:sim,2]),type="l",col=2,lwd=1)

lines(density(fit4$par[burnin:sim,2]),type="l",col=1,lwd=1)

legend("topleft",c("normal","rw","inde","gibbs"),

lty=1,col=c(4,3,2,1))



接下来，我们仿照table6.2做出四种算法的对比，首先就的0.05/0.5/0.95分位数而言，随机游走，独立模型和gibbs采样三种方法区别不大，都要好于正态。

> r12=rnorm(sim,fit$mode[2],sqrt(fit$var[2,2]))#normal

> r22=fit2$par[burnin:sim,2]#rw

> r32=fit3$par[burnin:sim,2]#inde

> r42=fit4$par[burnin:sim,2]#gibbs

>

> t12=c(quantile(r12,0.05),quantile(r12,0.5),quantile(r12,0.95))

> t22=c(quantile(r22,0.05),quantile(r22,0.5),quantile(r22,0.95))

> t32=c(quantile(r32,0.05),quantile(r32,0.5),quantile(r32,0.95))

> t42=c(quantile(r42,0.05),quantile(r42,0.5),quantile(r42,0.95))

> t12 #normal

5% 50% 95%

-0.10950724 -0.08203023 -0.05477066

> t22 #rw

5% 50% 95%

-0.11316273 -0.08419039 -0.05656357

> t32 #inde

5% 50% 95%

-0.11261074 -0.08422269 -0.05726191

> t42#gibbs

5% 50% 95%

-0.11345120 -0.08387958 -0.05673468

其次，对于的0.05/0.5/0.95分位数而言，四种方法相差极小，说明其有较好的正态性。

> r12=rnorm(sim,fit$mode[1],sqrt(fit$var[1,1])) #normal

> r22=fit2$par[burnin:sim,1] #rw

> r32=fit3$par[burnin:sim,1] #inde

> r42=fit4$par[burnin:sim,1] #gibbs

>

> t11=c(quantile(r12,0.05),quantile(r12,0.5),quantile(r12,0.95))

> t21=c(quantile(r22,0.05),quantile(r22,0.5),quantile(r22,0.95))

> t31=c(quantile(r32,0.05),quantile(r32,0.5),quantile(r32,0.95))

> t41=c(quantile(r42,0.05),quantile(r42,0.5),quantile(r42,0.95))

> t11#normal

5% 50% 95%

2.550118 2.795169 3.041423

> t21#rw

5% 50% 95%

2.550140 2.802765 3.035306

> t31 #inde

5% 50% 95%

2.555045 2.799454 3.041913

> t41#gibbs

5% 50% 95%

2.549682 2.805797 3.057397

## MAP

在本题中，由于后验概率形式比较简单，我们其实可以直接采用极大化后验概率(MAP)的方法求出最优的参数。即：

我们调用了GA包里的遗传算法对后验概率函数进行极大化求解。

library(GA)

f\_ga=function(beta0,beta1){

data=c(15,11,14,17,5,11,10,4,8,10,7,9,11,3,6,1,1,4)

i=1:18

y=data

sum((beta0+beta1\*i)\*y-exp(beta0+beta1\*i))

}

GA=ga(type = "real-valued",

fitness = function(x) f\_ga(x[1], x[2]),

lower = c(1, -2), upper = c(8, 1),

popSize = 50, maxiter = 1000, run = 100)

summary(GA)

经过611次迭代后得到最优解，2.82466， 这个结果与我们模拟得到的结果基本相同，相互佐证，说明的模拟方法与MAP的方法都具有很好的效果。

GA results:

Iterations = 611

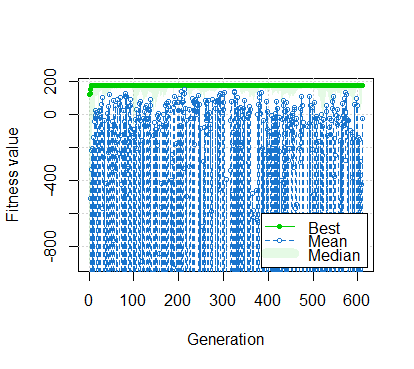
Fitness function value = 174.8327

Solution =

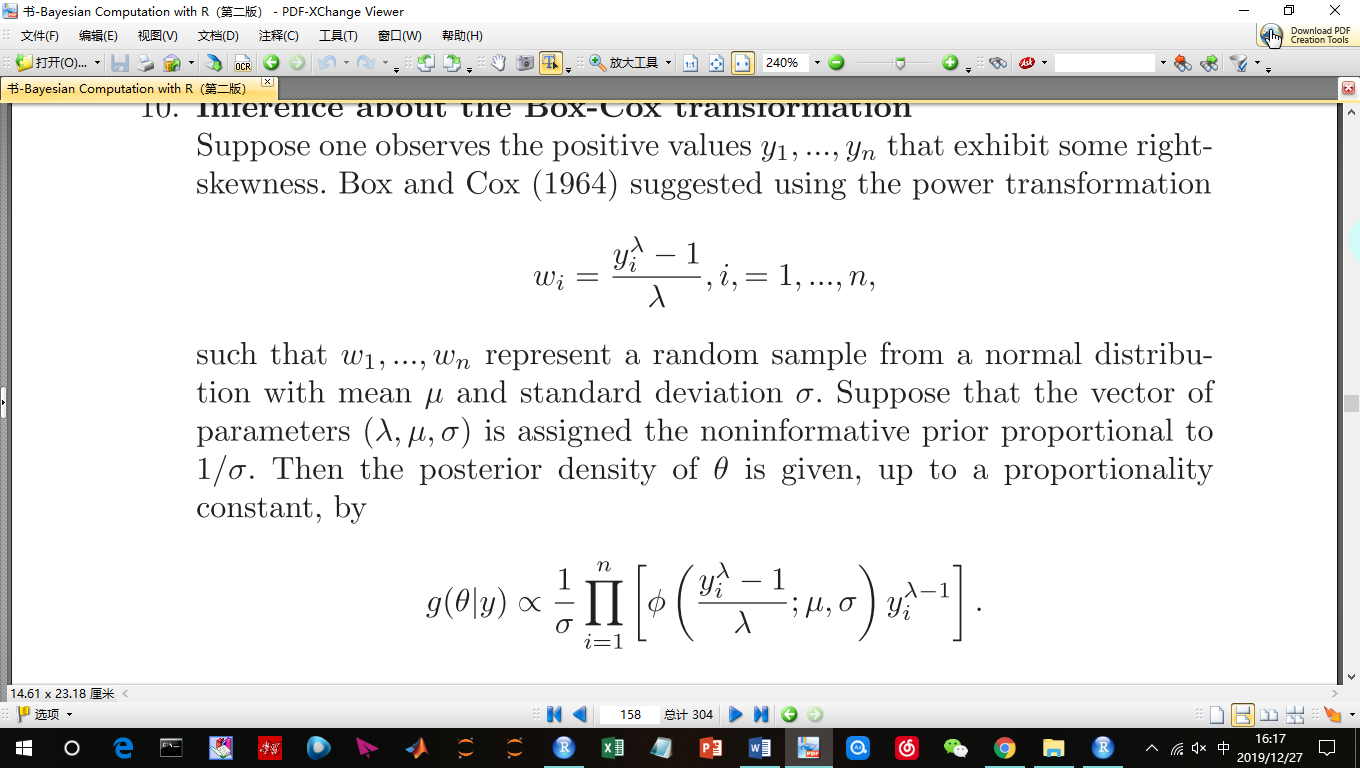
x1 x2

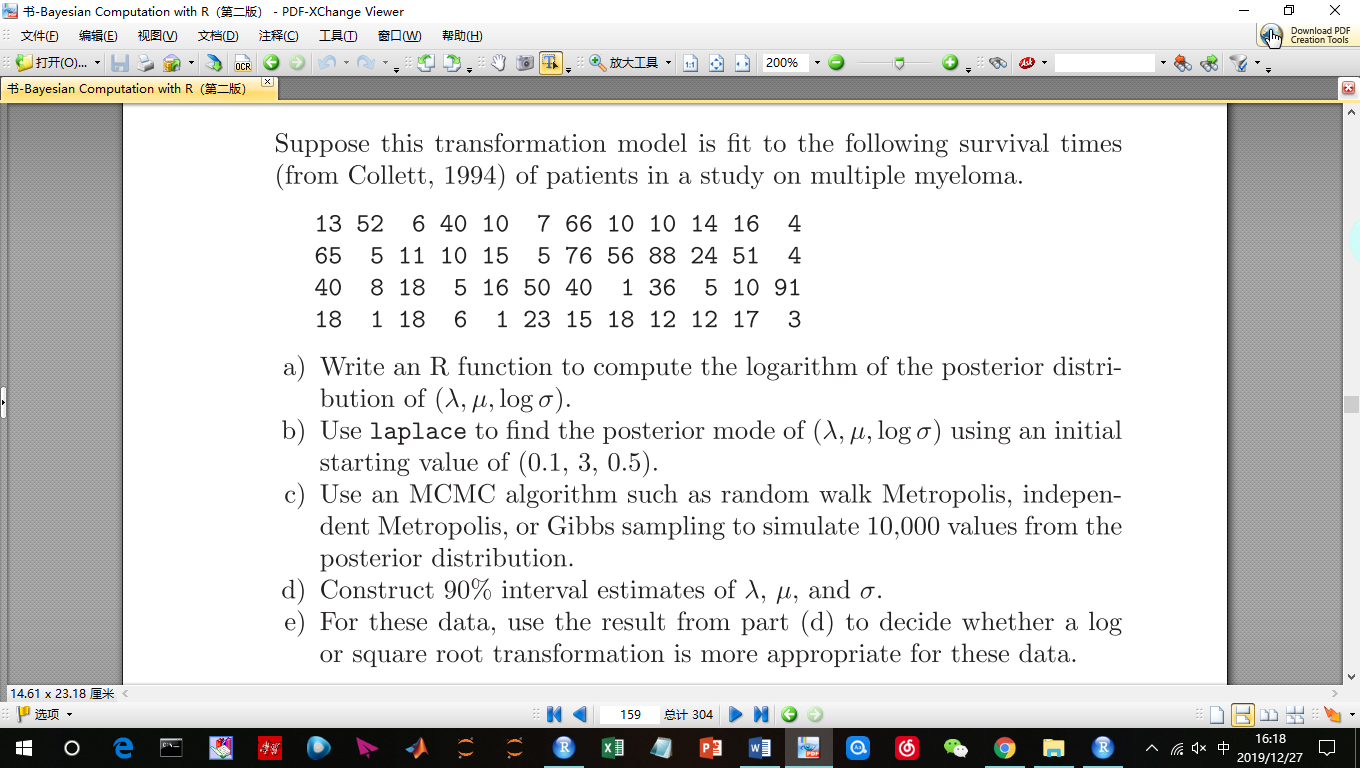
[1,] 2.82466 -0.08636227

plot(GA)



# Inference about the Box-Cox transformation(6.10)





本题，我们基于一组数据，利用Box-Cox方法对其做正态性转化。首先，题目给出了后验概率，我们需要写出log后验，并进行变量代换。

## 2.1 a

根据题意，我们写出函数计算上述log后验概率：

#6.10a

rm(list=ls())

set.seed(11)

library(LearnBayes)

data=c(13,52,6,40,10,7,66,10,10,14,16,4,65,5,11,10,15,5,76,

56,88,24,51,4,40,8,18,5,16,50,40,1,36,5,10,91,18,1,

18,6,1,23,15,18,12,12,17,3)

g=function(theta,data){

lambda=theta[1]

mu=theta[2]

sigma=exp(theta[3])

y=data

sum(log(dnorm((y^lambda-1)/lambda,mu,sigma))+(lambda-1)\*log(y))

}

## 2.2 b

接下来，使用laplace函数找到后验mode：

> start=c(0.1,3,0.5)

> laplacefit=laplace(g,start,data)

> laplacefit$mode

[1] 0.1371790 3.2757185 0.4565021

## 2.3 c

在此，我们使用三种MCMC算法进行模拟。此处由于估计的第三个参数为，为了得到真实值，我们需要取其指数。其接受率我们都控制在合理的范围内。其中gibbs方法的调参很麻烦，不太容易找到好的参数。

#rw

> proposal1=list(var=laplacefit$var,scale=1.7)

> fitrw=rwmetrop(g,proposal1,start,10000,data)

> fitrw$accept

[1] 0.2341

> pararw=fitrw$par

> pararw[,3]=exp(pararw[,3])

#inde

proposal2=list(var = laplacefit$var, mu = t(laplacefit$mode))

> fitinde=indepmetrop(g, proposal2, start, 10000, data)

> fitinde$accept

[,1]

[1,] 0.681

> parainde=fitinde$par

> parainde[,3]=exp(parainde[,3])

#gibbs

> fitgibbs=gibbs(g,start,10000,scale =c(0.15,1.5,0.55), data)

> fitgibbs$accept

[,1] [,2] [,3]

[1,] 0.221 0.2424 0.2272

> paragibbs=fitgibbs$par

> paragibbs[,3]=exp(paragibbs[,3])

用随机游走的模拟结果做展示：

|  |  |  |
| --- | --- | --- |
|  |  |  |

## 2.4 d

三种方法构建的90%置信区间：

> apply(pararw,2,quantile,c(.05,.95))

[,1] [,2] [,3]

5% -0.00628621 2.536748 1.109721

95% 0.39289331 5.383972 3.390467

> apply(parainde,2,quantile,c(.05,.95))

[,1] [,2] [,3]

5% -0.01250405 2.504812 1.070034

95% 0.35470770 4.801247 3.039811

> apply(paragibbs,2,quantile,c(.05,.95))

[,1] [,2] [,3]

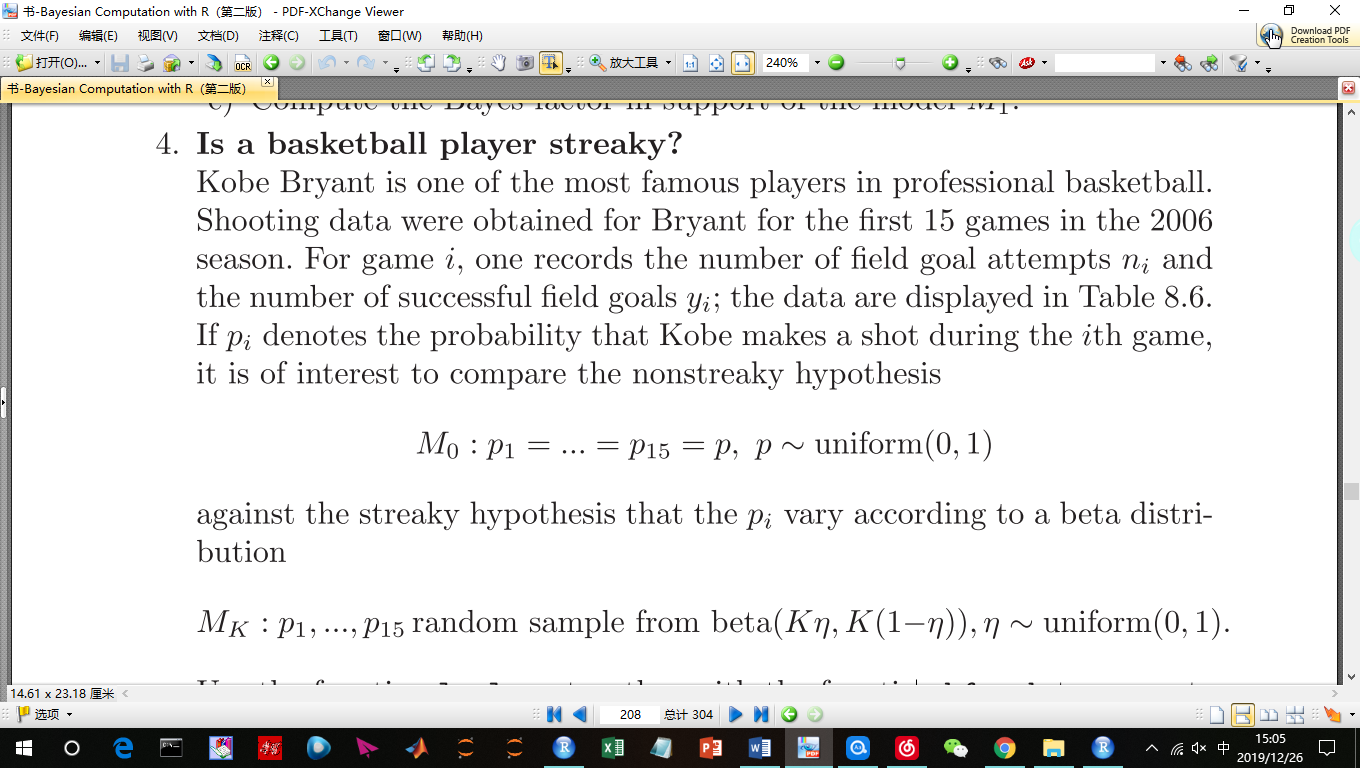
5% 0.0010025 2.563410 1.109809

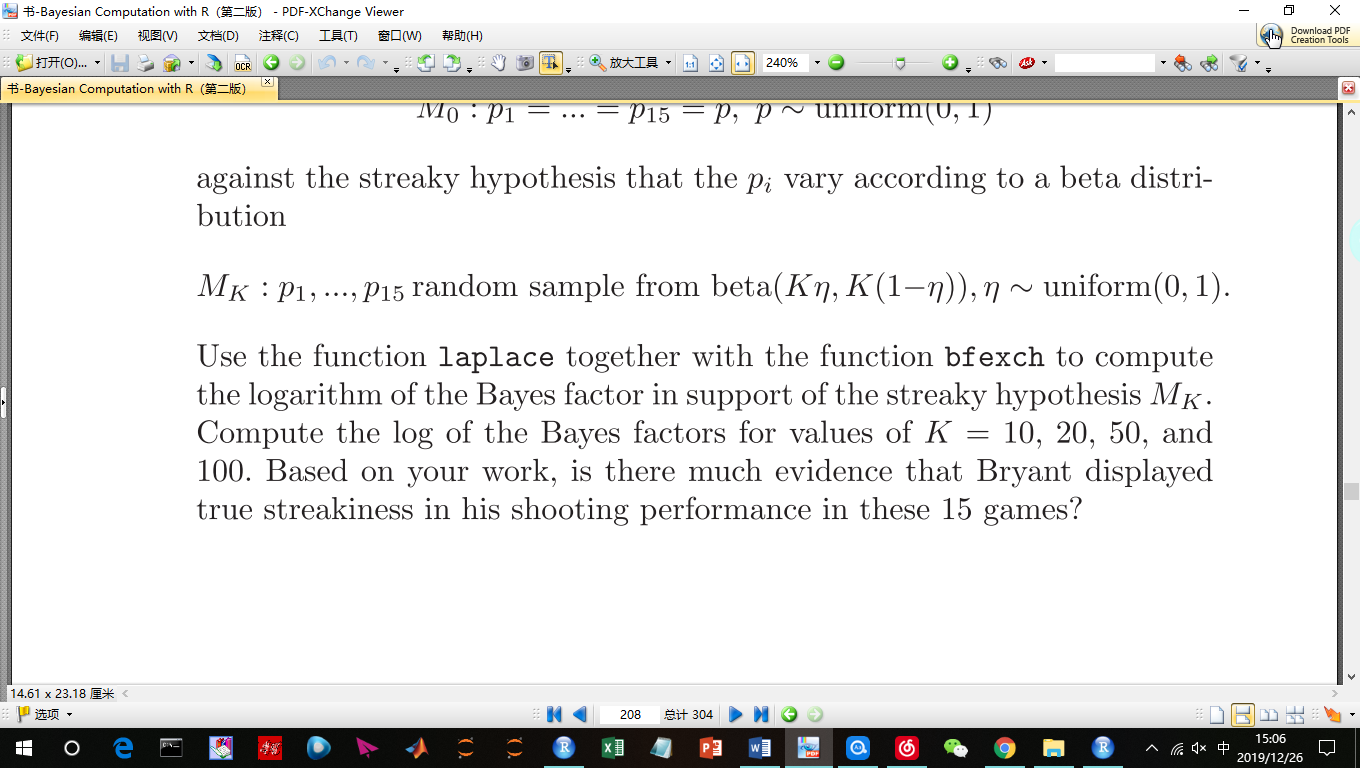
95% 0.4288908 5.704552 3.821411

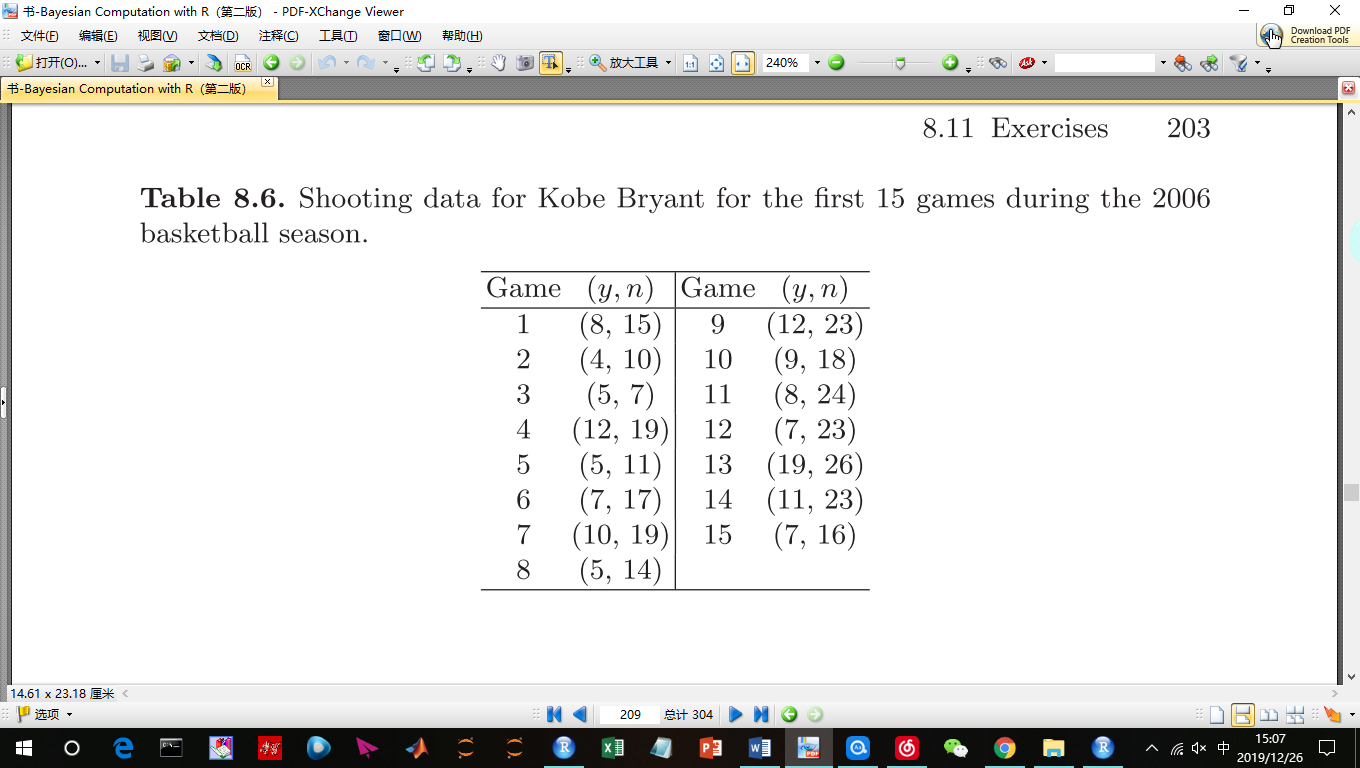
## 2.5 e

经过查阅资料我们发现当lambda=0时Box-Cox方法为log变换，通过随机游走和独立性采样得到的后验置信区间里都包含有0，所以我们认为，对于这组数据做log变换更加合适。

# Is a basketball player streaky? (8.4)







本题目，我们来判断Kobe投球的命中率是否遵循非条纹/条纹假设，我们仿照8.7节对baseball的做法。由于两个题目的基本问题是一致的，我们采取8.7节推导出的结果。

我们先构建了bfexch函数来计算贝叶斯因子：

bfexch=function (theta, datapar)

{

y = datapar$data[, 1]

n = datapar$data[, 2]

K = datapar$K

eta = exp(theta)/(1 + exp(theta))

logf = function(K, eta, y, n)

lbeta(K \* eta + y, K \* (1 - eta) + n - y) -

lbeta(K \* eta, K \* (1 - eta))

sum(logf(K, eta, y, n)) + log(eta \* (1 - eta)) -

lbeta(sum(y) + 1, sum(n - y) + 1)

}

导入数据，设置本题的精度参数K：

data=cbind(c(8,4,5,12,5,7,10,5,12,9,8,7,19,11,7),

c(15,10,7,19,11,17,19,14,23,18,24,23,26,23,16))

k=c(10,20,50,100)

为了计算特定K值的贝叶斯因子，我们使用函数laplace，输入函数bfexch，起始值η= 0。为了方便使用将其封装为函数。

log.marg=function(k){

laplace(bfexch,0,list(data=data,K=k))$int

}

使用sapply函数，我们计算不同K值下的log.BF，之后计算出贝叶斯因子的值，最后输出展示：

> log.BF=sapply(k,log.marg)

> BF=exp(log.BF)

> round(data.frame(k,log.BF,BF),2)

k log.BF BF

1 10 -1.42 0.24

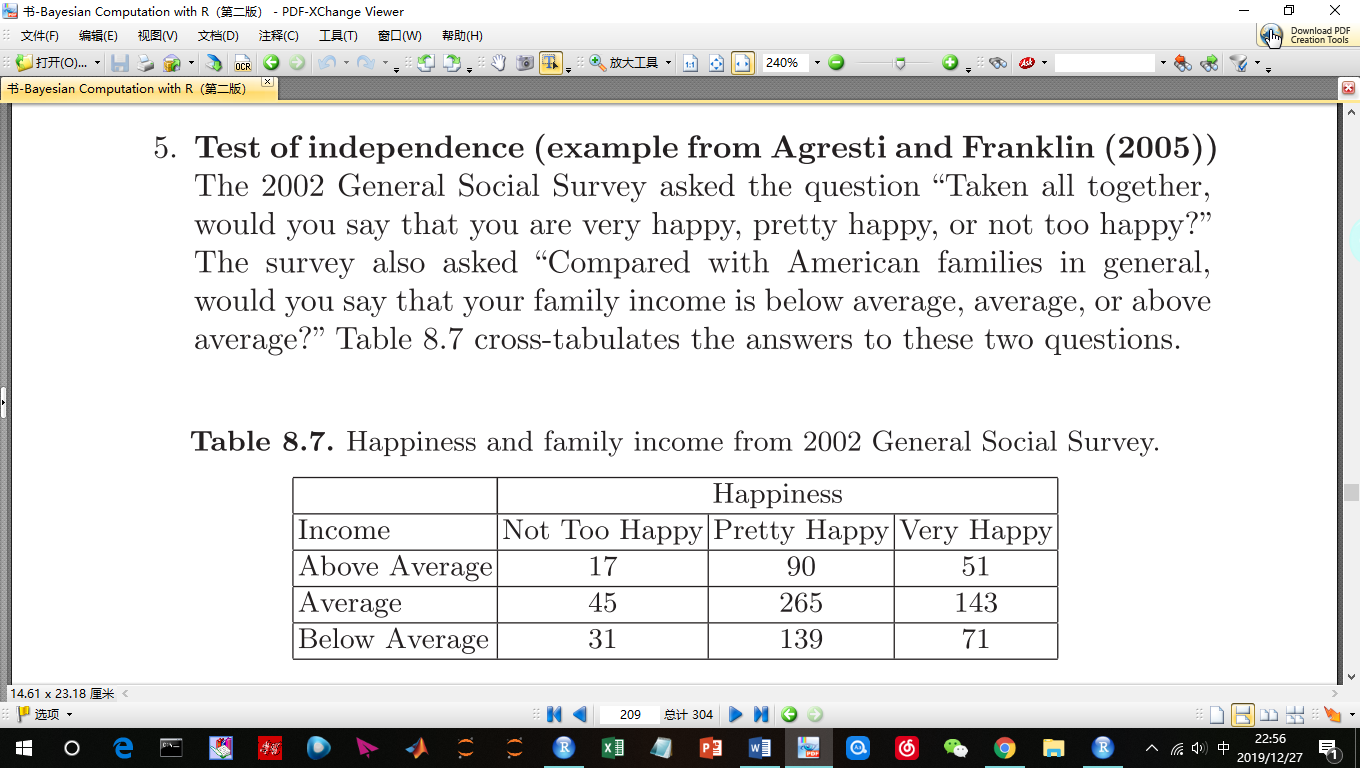
2 20 -0.09 0.91

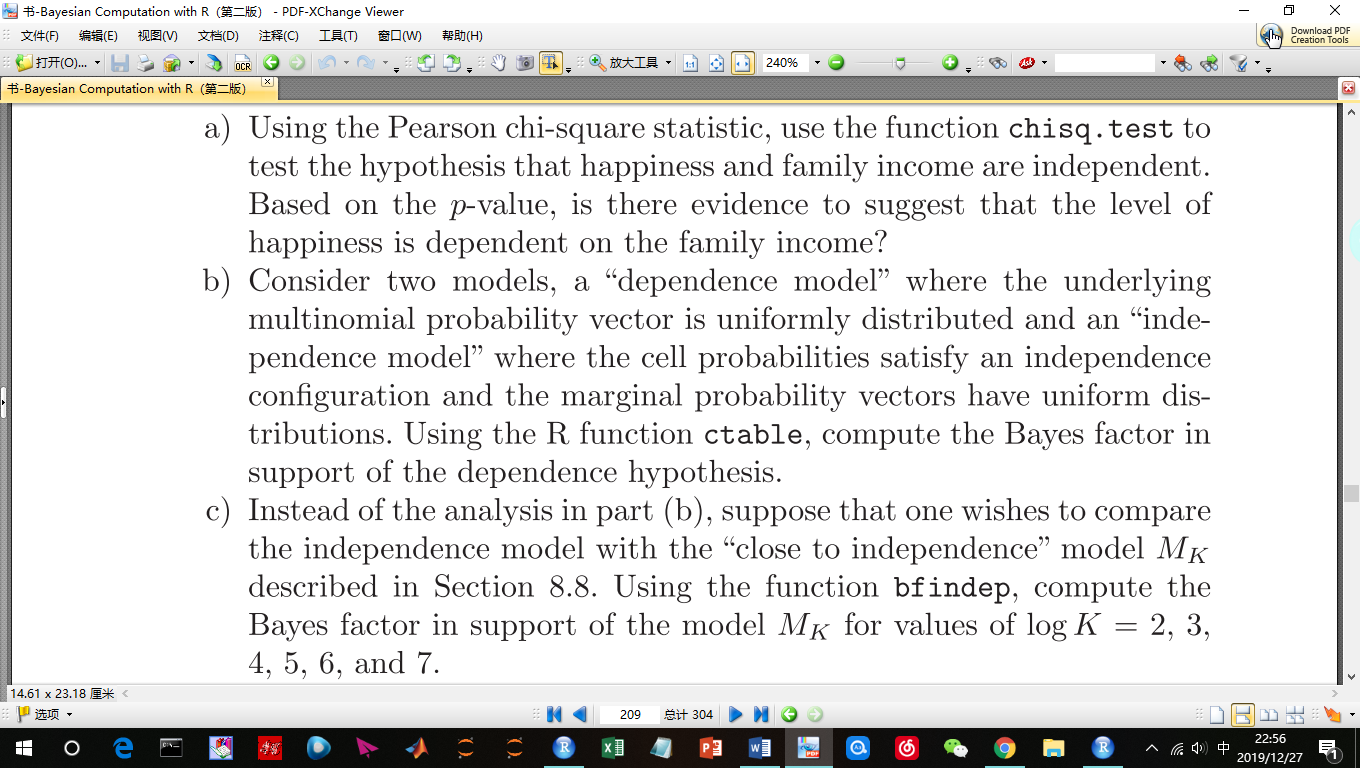
3 50 0.39 1.48

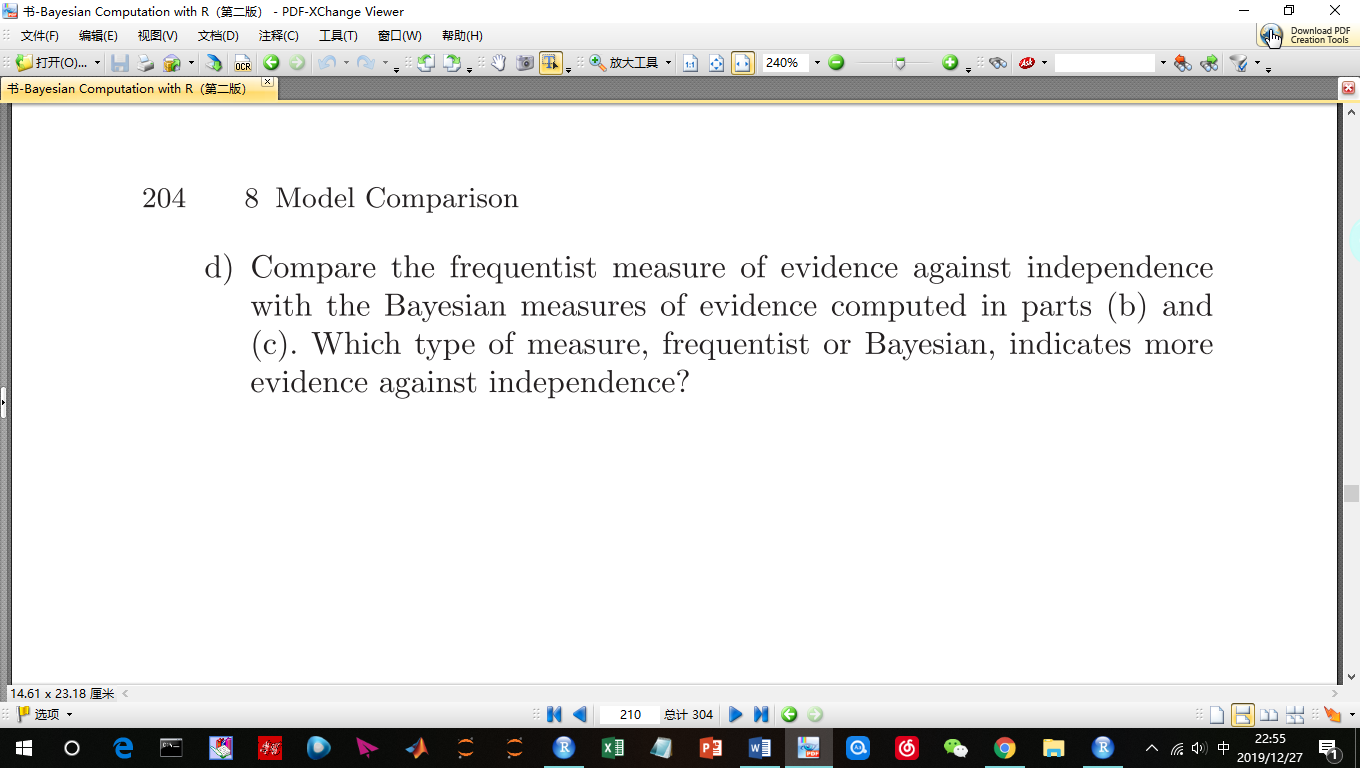
4 100 0.34 1.40

从输出结果我们可以看出，K=50时贝叶斯因子值最大，并且该条纹模型几乎是一致性模型的1.5倍。这表明了Kobe确实表现出了一种条纹模型的特性。

# Test of independence (example from Agresti and Franklin (2005)) (8.5)







## 4.1 a

第一问让我们对该列表做列联表独立性检验。我们仿照8.8节，先构造列联表，之后使用函数chisq.test求出相关性系数。P值为0.808，可以看出幸福与收入的相关性不显著。

> rm(list=ls())

> data=matrix(c(17,90,51,45,265,143,31,139,71),c(3,3),byrow = T)

> data

[,1] [,2] [,3]

[1,] 17 90 51

[2,] 45 265 143

[3,] 31 139 71

> chisq.test(data)

Pearson's Chi-squared test

data: data

X-squared = 1.6047, df = 4, p-value = 0.808

## 4.2 b

仿照8.8节，我们可以看出bayes因子的值仅为0.00066，说明dependence model更加符合实际。

> a=matrix(rep(1,9),c(3,3))

> a

[,1] [,2] [,3]

[1,] 1 1 1

[2,] 1 1 1

[3,] 1 1 1

> ctable(data, a)

[1] 0.0006591673

## 4.3 c

同样，仿照8.8节，我们根据不同的迪利克雷精度参数K计算相应的贝叶斯因子。可以看到在本例中最大的bayes因子为0.43，当logk=7时，说明此取值下，“close to independence”模型最符合实际。

> log.K=seq(2,7)

> compute.log.BF=function(log.K)log(bfindep(data,exp(log.K),100000)$bf)

> log.BF=sapply(log.K,compute.log.BF)

> BF=exp(log.BF)

> round(data.frame(log.K,log.BF,BF),2)

log.K log.BF BF

1 2 -15.06 0.00

2 3 -10.61 0.00

3 4 -6.49 0.00

4 5 -3.41 0.03

5 6 -1.83 0.16

6 7 -0.83 0.43