

**Lecture 9: Markov Chain Monte Carlo Methods**  
**(二)**  
**(马尔科夫蒙特卡罗方法)**

张伟平

Monday 2<sup>nd</sup> November, 2009

# Contents

<b>1</b>	<b>Markov Chain Monte Carlo Methods</b>	<b>1</b>
1.4	The Gibbs Sampler . . . . .	1
1.4.1	The Slice Gibbs Sampler . . . . .	12
1.5	Monitoring Convergence . . . . .	21
1.5.1	Convergence diagnostics plots . . . . .	21
1.5.2	Monte Carlo Error . . . . .	22
1.5.3	The Gelman-Rubin Method . . . . .	25
1.6	WinBUGS Introduction . . . . .	33
1.6.1	Building Bayesian models in WinBUGS . . . . .	33
1.6.2	Model specification in WinBUGS . . . . .	36
1.6.3	Data and initial value specification . . . . .	38
1.6.4	Compiling model and simulating values . . . . .	46

# Chapter 1

## Markov Chain Monte Carlo Methods

### 1.4 The Gibbs Sampler

Gibbs 抽样是MH算法的一个特例, 其经常用于目标分布是多元分布的场合. 假设所有的一元条件分布 (每个分量对其他分量的条件分布)都是可以确定的, Gibbs抽样使用这些一元条件分布进行抽样.

令  $X = (X_1, \dots, X_d)$  为  $R^d$  中的随机变量, 定义  $d - 1$  维的随机变量

$$X_{-j} = (X_1, \dots, X_{j-1}, X_{j+1}, \dots, X_d),$$

并记  $X_j|X_{-j}$  的条件密度为  $f(X_j|X_{-j})$ . 则Gibbs抽样是从这  $d$  个条件分布中产生候选点. 算法如下:

1. 在  $t = 0$  时, 初始化  $X(0)$ ;

2. 对  $t = 1, 2, \dots, T$ ,

(a) 令  $x_1 = X_1(t-1)$ .

(b) 对每个分量  $j = 1, \dots, d$ ,

(i) 从  $f(X_j|x_{-j})$  中产生候选点  $X_j^*(t)$ .

(ii) 更新  $x_j = X_j^*(t)$ .

(c) 令  $X(t) = (X_1^*(t), \dots, X_d^*(t))$  (每个候选点都被接受)

(d) 增加  $t$

注意在上述算法(b)步抽样中, 各个分量依次被更新:

$$x_1(t) \sim f(x_1|x_2(t-1), \dots, x_d(t-1));$$

$$x_2(t) \sim f(x_2|x_1(t), x_3(t-1), \dots, x_d(t-1))$$

$\vdots$

$$x_d(t) \sim f(x_d|x_1(t), \dots, x_{d-1}(t)).$$

从一元分布 $f(x_j|x_1(t), x_2(t), \dots, x_{j-1}(t), x_{j+1}(t-1), \dots, x_d(t-1))$ 中抽样是比较容易的, 因为 $f(x_j|x_{-j}) \propto f(x)$ , 其中除了变量 $x_j$ 外, 其他变量都是常数.

**例 1 (Gibbs 抽样: 二元分布)** 使用Gibbs抽样产生二元正态分布 $N(\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho)$  的随机数

在二元正态场合,  $X_1|X_2$ 以及 $X_2|X_1$ 仍然服从正态分布, 且易知

$$E[X_1|X_2 = x_2] = \mu_1 + \rho \frac{\sigma_1}{\sigma_2} (x_2 - \mu_2),$$
$$\text{Var}[X_1|X_2 = x_2] = (1 - \rho^2) \sigma_1^2$$

类似可得 $X_2|X_1$ 的分布. 因此

$$f(x_1|x_2) \sim N\left(\mu_1 + \rho \frac{\sigma_1}{\sigma_2} (x_2 - \mu_2), (1 - \rho^2) \sigma_1^2\right)$$
$$f(x_2|x_1) \sim N\left(\mu_2 + \rho \frac{\sigma_2}{\sigma_1} (x_1 - \mu_1), (1 - \rho^2) \sigma_2^2\right)$$

因此, 使用Gibbs算法如下

1. 令 $(x_1, x_2) = X(t - 1)$ ;
2. 从 $f(x_1|x_2)$ 中产生候选点 $X_1^*(t)$ .
3. 更新 $x_1 = X_1^*(t)$ .
4. 从 $f(x_2|x_1)$ 中产生 $X_2^*(t)$ .
5. 令 $X(t) = (X_1^*(t), X_2^*(t))$ .

R 代码如下

```
#initialize constants and parameters
N <- 5000                #length of chain
burn<- 1000             #burn-in length
X <- matrix(0, N, 2)    #the chain, a bivariate sample

rho <- -.75              #correlation
mu1 <- 0
mu2 <- 2
sigma1 <- 1
sigma2 <- .5
s1 <- sqrt(1-rho^2)*sigma1
```

[↑Code](#)

```
s2 <- sqrt(1-rho^2)*sigma2

##### generate the chain #####

X[1, ] <- c(mu1, mu2)           #initialize

for (i in 2:N) {
  x2 <- X[i-1, 2]
  m1 <- mu1 + rho * (x2 - mu2) * sigma1/sigma2
  X[i, 1] <- rnorm(1, m1, s1)
  x1 <- X[i, 1]
  m2 <- mu2 + rho * (x1 - mu1) * sigma2/sigma1
  X[i, 2] <- rnorm(1, m2, s2)
}

b <- burn + 1
x <- X[b:N, ]
```

[↓Code](#)

---

产生的链开始的1000个观测被丢弃掉，剩下的观测存在 $x$ 中，对此样本 计算均值和协方差矩阵如下。各参数的样本估计离真值很近，散点图也显示出 二元正

态所具有的球面对称性和负相关性特征.

```
# compare sample statistics to parameters
colMeans(x)
cov(x)
cor(x)

plot(x, main="", cex=.5, xlab=bquote(X[1]),
      ylab=bquote(X[2]), ylim=range(x[,2]))
```

↑Code

↓Code

例 2 (贝叶斯分析例子: 身体温度数据) 考虑 Mackowiak et al. (1992)的数据, 该数据记录了 130个人的身体温度(华氏), 性别和每分钟的心跳. 实验的目的是检验Carl Wunderlich的观点— 健康成年人的体温平均为 $37^{\circ}C (=98.6^{\circ}F)$ .



记温度为 $y_i, i = 1, \dots, n$ , 并假设正态模型

$$y_i \sim N(\mu, \sigma^2)$$

以及取先验分布为

$$\mu \sim N(\mu_0, \sigma_0), \quad \sigma^2 \sim IG(a_0, b_0)$$

则此时我们的目标分布为 $\mu, \sigma^2$ 的后验分布

$$f(\mu, \sigma^2 | y) \propto f(y | \mu, \sigma^2) \pi(\mu) \pi(\sigma^2)$$

为使用Gibbs抽样算法, 我们必须计算 $f(\mu | \sigma^2, y)$ 与 $f(\sigma^2 | \mu, y)$ . 经过计算我们得到

$$\mu | \sigma, y \sim N(\omega \bar{y} + (1 - \omega) \mu_0, \omega \frac{\sigma^2}{n}), \omega = \frac{\sigma_0^2}{\sigma^2/n + \sigma_0^2}.$$

$$\sigma^2 | \mu, y \sim IG(a_0 + \frac{n}{2}, b_0 + \frac{1}{2} \sum_{i=1}^n (y_i - \mu)^2).$$

使用此结果, Gibbs抽样算法如下:

对  $t = 1, \dots, T$ ,

1. 令  $\mu = \mu^{(t-1)}, \sigma = \sigma^{(t-1)}$ .
2. 计算  $\omega = \frac{\sigma_0^2}{\sigma^2/n + \sigma_0^2}$ ,  $m = \omega \bar{y} + (1 - \omega)\mu_0$  和  $s^2 = \omega \frac{\sigma^2}{n}$ .
3. 从  $N(m, s^2)$  中产生  $\mu$ .
4. 令  $\mu^{(t)} = \mu$ .
5. 计算  $a = a_0 + \frac{n}{2}$ ,  $b = b_0 + \frac{1}{2} \sum_{i=1}^n (y_i - \mu)^2$ .
6. 从  $G(a, b)$  中产生  $\tau$ .
7. 令  $\sigma^2 = 1/\tau$  以及  $\sigma^{(t)} = \sigma$ .

从而R代码如下:

```
bodytemp<-read.table("bodytemp.txt",header=T)
y<-bodytemp$temp
bary<-mean(y); n<-length(y)
Iterations<-3500
mu0<-0; s0<-100; a0<-0.001; b0<-0.001
```

[↑Code](#)

```
theta <- matrix(nrow=Iterations, ncol=2)
cur.mu<-0; cur.tau<-2; cur.s<-sqrt(1/cur.tau)
for (t in 1:Iterations){
  w<- s0^2/( cur.s^2/n+ s0^2 )
  m <- w*bary + (1-w)*mu0
  s <- sqrt( w/n ) * cur.s
  cur.mu <- rnorm( 1, m, s )
  a <- a0 + 0.5*n
  b <- b0 + 0.5 * sum( (y-cur.mu)^2 )
  cur.tau <- rgamma( 1, a, b )
  cur.s <- sqrt(1/cur.tau)
  theta[t,]<-c( cur.mu, cur.s)
}
mcmc.output<-theta
apply(mcmc.output[-(1:1000),],2,mean)
#compare to true value: 98.25, 0.542
apply(mcmc.output[-(1:1000),],2,sd)
#compare to true value: 0.06456, 0.06826
```

[↓ Code](#)

## 对链的诊断图

[↑Code](#)

```
par( mfrow=c(3,2), xaxs='r', yaxs='r', bty='l' , cex=0.8)
iter<-1500
burnin<-500
index<-1:iter
index2<-(burnin+1):iter

plot(index, theta[index,1], type='l', ylab='Values of mu',
      xlab='Iterations', main='(a) Trace Plot of mu')
plot(index, theta[index,2], type='l', ylab='Values of sigma',
      xlab='Iterations', main='(b) Trace Plot of sigma')

ergtheta0<-erg.mean( theta[index,1] )
ergtheta02<-erg.mean( theta[index2,1] )
ylims0<-range( c(ergtheta0,ergtheta02) )

ergtheta1<-erg.mean( theta[index,2] )
ergtheta12<-erg.mean( theta[index2,2] )
ylims1<-range( c(ergtheta1,ergtheta12) )
```

```
step<-10
index3<-seq(1,iter,step)
index4<-seq(burnin+1,iter,step)

plot(index3 , ergtheta0[index3], type='l', ylab='Values of mu',
xlab='Iterations', main='(c) Ergodic Mean Plot of mu', ylim=ylimits0)
lines(index4, ergtheta02[index4-burnin], col=2, lty=2)

plot(index3, ergtheta1[index3], type='l', ylab='Values of sigma',
xlab='Iterations', main='(d) Ergodic Mean Plot of sigma', ylim=ylimits1)
lines(index4, ergtheta12[index4-burnin], col=2, lty=2)

acf(theta[index2,1], main='Autocorrelations Plot for mu')
acf(theta[index2,2], main='Autocorrelations Plot for sigma')
```

[↓Code](#)

## 1.4.1 The Slice Gibbs Sampler

切片Gibbs抽样(Slice Gibbs Sampler) 本质上是基于Gibbs抽样的. 其主要用于当 完全的条件分布没有简单或者方便的形式情形. 这个方法通过添加一些辅助变量 把参数空间扩大, 但保持感兴趣的边际分布不变, 而把所有的条件分布转变成标准形式. 然后可以使用标准的Gibbs抽样方法.

切片Gibbs抽样的想法如下. 考虑目标分布 $g(x)$ , 其很难进行抽样. 我们引入一个新的辅助变量 $u$ , 其条件分布为 $f(u|x)$ . 则联合分布为

$$f(u, x) = f(u|x)g(x)$$

而 $x$ 的边际分布等于目标分布 $g(x)$ . 因此我们可以使用Gibbs算法从联合分布 $f(u, x)$ , 以及边际分布 $f(u)$ 和 $f(x) = g(x)$ 中产生随机数:

1. 产生 $u \sim f(u|x)$ .
2. 产生 $x \sim f(u|x)g(x)$ .

由于 $f(u|x)$ 在上述计算中出现两次, 因此其的选取要使得从分布 $f(u|x)$ 和 $f(u|x)g(x)$

中很方便的抽样, 常用的一个选择是均匀分布 $U(0, g(x))$ , 此时

$$f(u, x) = \frac{1}{g(x)}g(x)I(0 < u < g(x)) = I(0 < u < g(x)),$$

$$f(x) = \int I(0 < u < g(x))du = g(x).$$

因此, 此时Gibbs抽样算法如下

1. 产生 $u^{(t)} \sim U(0, g(x^{(t-1)}))$ ;
2. 产生 $x^{(t)} \sim U(x : 0 \leq u^{(t)} \leq g(x))$ .

对贝叶斯分析来说, 经常选择 $u \sim U(0, f(y|\theta))$ , 联合分布为

$$f(\theta, u|y) \propto \left\{ \prod_{i=1}^n I(0 \leq u_i \leq f(y_i|\theta)) \right\} f(\theta)$$

从而Gibbs算法如下

1. 令 $\theta = \theta^{(t-1)}$ .
2. 对 $i = 1, \dots, n$ , 产生 $u_i^{(t)} \sim U(0, f(y_i|\theta))$ .
3. 对 $j = 1, \dots, d$ , 更新 $\theta_j \sim f(\theta_j) \prod_{i=1}^n I(0 \leq u_i^{(t)} \leq f(y_i|\theta))$ .

4. 令 $\theta^{(t)} = \theta$ .

**例 3 (切片Gibbs抽样: logistic回归中的应用)** 考虑WAIS数据分析的例子.

我们选取辅助变量使得

$$f(u, \beta_0, \beta_1 | y) \propto \prod_{i=1}^n I(u_i \leq \frac{e^{\beta_0 y_i + \beta_1 x_i y_i}}{1 + e^{\beta_0 + \beta_1 x_i}}) \exp\left(-\frac{(\beta_0 - \mu_{\beta_0})^2}{2\sigma_{\beta_0}^2} - \frac{(\beta_1 - \mu_{\beta_1})^2}{2\sigma_{\beta_1}^2}\right)$$

$\beta_0, \beta_1$ 的边际分布为

$$\begin{aligned} f(\beta_0, \beta_1 | y) &= \int f(u, \beta_0, \beta_1 | y) du \\ &\propto \prod_{i=1}^n \frac{e^{\beta_0 y_i + \beta_1 x_i y_i}}{1 + e^{\beta_0 + x_i \beta_1}} \exp\left(-\frac{(\beta_0 - \mu_{\beta_0})^2}{2\sigma_0^2} - \frac{(\beta_1 - \mu_{\beta_1})^2}{2\sigma_1^2}\right). \end{aligned}$$

即为我们在第8讲例7中的模型. 因此我们使用切片Gibbs抽样方法

1. 对 $i = 1, \dots, n$ , 从如下分布中产生 $u_i$ ,

$$u_i | u_{-i}, \beta_0, \beta_1, y \sim U\left(0, \frac{e^{\beta_0 y_i + \beta_1 x_i y_i}}{1 + e^{\beta_0 + x_i \beta_1}}\right)$$



2. 从如下分布中产生 $\beta_0$ ,

$$\beta_0 | u, \beta_1, y \sim N(\mu_{\beta_0}, \sigma_{\beta_0}^2) \prod_{i=1}^n I(u_i \leq \frac{e^{\beta_0 y_i + \beta_1 x_i y_i}}{1 + e^{\beta_0 + x_i \beta_1}}),$$

3. 从如下分布中产生 $\beta_1$ ,

$$\beta_1 | u, \beta_0, y \sim N(\mu_{\beta_1}, \sigma_{\beta_1}^2) \prod_{i=1}^n I(u_i \leq \frac{e^{\beta_0 y_i + \beta_1 x_i y_i}}{1 + e^{\beta_0 + x_i \beta_1}}).$$

上述条件后验分布为截断的正态分布. 截断的区间定义为 $u_i \leq \frac{e^{\beta_0 y_i + \beta_1 x_i y_i}}{1 + e^{\beta_0 + x_i \beta_1}}$ , 其可以重新表示为

$$\text{For } y_i = 1 \implies u_i \leq \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + x_i \beta_1}} \implies \beta_0 + \beta_1 x_i \geq \log \frac{u_i}{1 - u_i}$$

$$\text{For } y_i = 0 \implies u_i \leq \frac{1}{1 + e^{\beta_0 + x_i \beta_1}} \implies \beta_0 + \beta_1 x_i \leq \log \frac{1 - u_i}{u_i}$$

因此得到

$$\max_{i: y_i=1} (\log \frac{u_i}{1 - u_i}) \leq \beta_0 + \beta_1 x_i \leq \min_{i: y_i=0} (\log \frac{1 - u_i}{u_i})$$

对 $\beta_0, \beta_1$ 解上述不等式得到

$$l_0 = \max_{i:y_i=1} \left( \log \frac{u_i}{1-u_i} - \beta_1 x_i \right) \leq \beta_0 \leq u_0 = \min_{i:y_i=0} \left( \log \frac{1-u_i}{u_i} - \beta_1 x_i \right)$$

以及

$$l_1 = \max_{i:y_i=1} \left( x_i^{-1} \left[ \log \frac{u_i}{1-u_i} - \beta_0 \right] \right) \leq \beta_1 \leq u_1 = \min_{i:y_i=0} \left( x_i^{-1} \left[ \log \frac{1-u_i}{u_i} - \beta_0 \right] \right)$$

在此例中,所有的 $x_i > 0$ . 因此参数 $\beta_0, \beta_1$ 最终由分布 $N(\mu_{\beta_0}, \sigma_{\beta_0}^2)I(l_0, u_0)$ 和 $N(\mu_{\beta_1}, \sigma_{\beta_1}^2)I(l_1, u_1)$ 产生.

这里我们要从一个截断分布中产生随机数, 这并不困难. 事实上, 若要从如下截断分布中抽样

$$F_{[a,b]}^T(x) = P(X \leq x | a \leq X \leq b) = \frac{F(x) - F(a)}{F(b) - F(a)}, \quad \forall x \in [a, b]$$

则我们可以产生 $u \sim U(0, 1)$ , 然后令 $u = F_{[a,b]}^T(x)$ , 那么解此方程得到

$$x = F^{-1}(F(a) + u[F(b) - F(a)]).$$

因此实现如上分析的R代码如下

[↑Code](#)

```
y<-wais$senility; x<-wais$wais; n<-length(y)
positive<- y==1
Iterations<-55000
mu.beta<-c(0,0); s.beta<-c(100,100)
beta <- matrix(nrow=Iterations, ncol=2)
acc.prob <- 0
current.beta<-c(0,0); u<-numeric(n)
for (t in 1:Iterations){
  eta<-current.beta[1]+current.beta[2]*x
  U<-exp(y*eta)/(1+exp(eta))
  u<-runif( n, rep(0,n), U)
  logitu<-log( u/(1-u) )
  logitu1<- logitu[positive]
  logitu2<- -logitu[!positive]

  l0<- max( logitu1 - current.beta[2]*x[positive] )
  u0<- min( logitu2 - current.beta[2]*x[!positive] )
  unif.random<-runif(1,0,1)
  fa<- pnorm(l0, mu.beta[1], s.beta[1])
}
```

```
fb<- pnorm(u0, mu.beta[1], s.beta[1])
current.beta[1] <- qnorm( fa + unif.random*(fb-fa),
                        mu.beta[1], s.beta[1])

l1<- max( (logitu1 - current.beta[1])/x[positive] )
u1<- min( (logitu2 - current.beta[1])/x[!positive] )
unif.random<-runif(1,0,1)
fa<- pnorm(l1, mu.beta[2], s.beta[2])
fb<- pnorm(u1, mu.beta[2], s.beta[2])
current.beta[2] <- qnorm( fa + unif.random*(fb-fa),
                        mu.beta[2], s.beta[2])

beta[t,]<-current.beta
}
apply(beta[-(1:15000),],2,mean)
apply(beta[-(1:15000),],2,sd)
```

[↓Code](#)

---

链的收敛诊断图为

```
par( mfrow=c(3,2), xaxs='r', yaxs='r', bty='l' , cex=0.8)
iter<-55000
```

[↑Code](#)

```
burnin<-15000
index<-seq(1,iter,50)
index2<-(burnin+1):iter

plot(index, beta[index,1], type='l', ylab='Values of beta0',
      xlab='Iterations', main='(a) Trace Plot of beta0')
plot(index, beta[index,2], type='l', ylab='Values of beta1',
      xlab='Iterations', main='(b) Trace Plot of beta1')

iter<-55000
burnin<-15000
index<-seq(1,iter,1)
index2<-(burnin+1):iter

ergbeta0<-erg.mean( beta[index,1] )
ergbeta02<-erg.mean( beta[index2,1] )
ylims0<-range( c(ergbeta0,ergbeta02) )

ergbeta1<-erg.mean( beta[index,2] )
ergbeta12<-erg.mean( beta[index2,2] )
ylims1<-range( c(ergbeta1,ergbeta12) )
```

```
step<-50
index3<-seq(1,iter,step)
index4<-seq(burnin+1,iter,step)

plot(index3 , ergbeta0[index3], type='l', ylab='Values of beta0',
      xlab='Iterations', main='(c) Ergodic Mean Plot of beta0', ylim=ylimits0)
lines(index4, ergbeta02[index4-burnin], col=2, lty=2)

plot(index3, ergbeta1[index3], type='l', ylab='Values of beta1',
      xlab='Iterations', main='(d) Ergodic Mean Plot of beta1', ylim=ylimits1)
lines(index4, ergbeta12[index4-burnin], col=2, lty=2)

lag.to.print<-900
acf1<-acf(beta[index2,1], main='Autocorrelations Plot for beta0',
          lag.max=lag.to.print, plot=FALSE)
acf2<-acf(beta[index2,2], main='Autocorrelations Plot for beta1',
```

```
lag.max=lag.to.print, plot=FALSE)

acf.index<-seq(1,lag.to.print,20)

plot( acf1[acf.index], main='Auto-correlations for beta0' )
plot( acf2[acf.index], main='Auto-correlations for beta1' )
```

[↓ Code](#)

## 1.5 Monitoring Convergence

检查产生的链是否收敛，最直接简单的就是图示了。

### 1.5.1 Convergence diagnostics plots

*trace plot*: 将所有生成的样本对迭代次数作图，生成了链的一条样本路径图。当链达到收敛时，此路径图就应该呈现出稳定性，没有明显的趋势和周期。

*ergodic mean plot*: MCMC方法的理论基础是遍历均值定理，因此可以监

视遍历 均值是否达到收敛. 我们可以使用累积均值对迭代次数作图, 以观察遍历均值是否达到收敛.

*Autocorrelation plot*: 由于模型中的参数一般是相关的, 因此Gibbs抽样走遍目标 分布整个支撑的速度一般就会比较慢. 如果自相关水平很高, 则使用*trace plot*诊断链 的收敛性就比较差. 一般自相关随着步长的增大而减少, 如果某个链没有表现出这种现象, 那么说明链的产生机制有问题, 可能需要重新参数化.

## 1.5.2 Monte Carlo Error

在对MCMC生产的链进行分析时, Monte Carlo 误差 (MC 误差) 是需要监视的一个指标, 其衡量了估计量由于随机模拟 而带来的波动性. 在计算感兴趣的参数时, Monte Carlo误差必须很低且精度可以随着样本量递增. 其 和产生的样本量成反比, 因此用户自己可以控制. 从而增加迭代次数, 感兴趣的量就可以以递增 的精度被估计.



有两种计算MC误差的方法: *batch mean*和*windows estimator*. 第一种方法简单容易操作, 但是第二种方法精确.

使用*batch mean*方法时, 首先将生成的 $T$ 个样本分成 $K$ 个组(batch), 每个组 $v = T/K$ 个,  $K$ 常取为30或50.  $v$ 和 $K$ 都要比较大, 以使得我们可以相合的估计方差以及减少自相关. 在计算估计量 $g(X)$ 的MC误差时, 首先计算每个组内均值

$$\overline{g(X)}_b = \frac{1}{v} \sum_{t=(b-1)v+1}^{bv} g(X^{(t)}), \quad b = 1, \dots, K.$$

以及总的样本均值

$$\overline{g(X)} = \frac{1}{K} \sum_{b=1}^K \overline{g(X)}_b.$$

因此均值的MC误差估计为

$$MCE(g(X)) = \hat{se}(\overline{g(X)}) = \sqrt{\frac{1}{K(K-1)} \sum_{b=1}^k (\overline{g(X)}_b - \overline{g(X)})^2}$$

MC误差的 *batch mean* 估计方法更多的讨论可以参见 Hastings (1970), Geyer (1992), Roberts (1996, p. 50), Carlin and Louis (2000, p. 172), and Givens and Hoeting (2005, p. 208).

*window estimator* 是基于Roberts (1996, p. 50)对自相关样本的样本方差表示

$$MCE(g(X)) = \frac{\hat{S}D(g(X))}{\sqrt{T}} \sqrt{1 + 2 \sum_{k=1}^{\infty} \hat{\rho}_k(g(X))},$$

其中  $\hat{\rho}_k g(X)$  估计  $g(X^{(t)})$  与  $g(X^{(t+k)})$  之间的  $k$  阶自相关系数. 很显然, 对很大的  $k$ , 自相关系数  $\hat{\rho}_k$  由于样本量很少而不能很好的估计, 而且对充分的  $k$ , 自相关将接近0. 因此, 取一个窗口  $\omega$  使得其后的自相关系数都很小, 在计算中就可以丢掉后面所有的值. 这种基于窗口的MC误差为

$$MCE(g(X)) = \frac{\hat{S}D(g(X))}{\sqrt{T}} \sqrt{1 + 2 \sum_{k=1}^{\omega} \hat{\rho}_k(g(X))},$$

### 1.5.3 The Gelman-Rubin Method

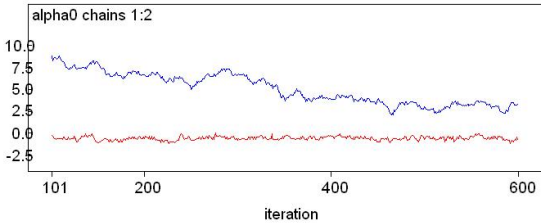
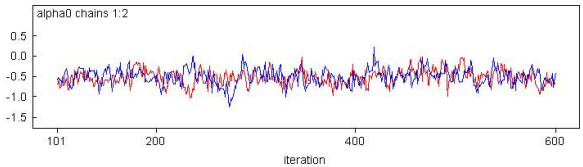
Gelman & Rubin <sup>1</sup> 给了一个例子, 说明 很慢的收敛不能通过单独检查一个链来发现. 单独一个链也许看起来已经收敛, 但是实际上在整个支撑上链没有达到收敛. 因为如果在目标分布的一个局部支撑上, 产生的值的方差非常小, 则就会出现这种情况. 因而 通过检查几个平行的链, 其初始值非常分散, 那么能发现收敛很慢的效率就会高得多. 这个方法是基于从不同的 初始值出发, 链达到平稳后, 应该表现的是一样的. 更准确的说, Gelman 和 Rubin 指出链内的方差和链 之间的方差应该是相同的.

这个想法可以通过将多个链的 *trace plot*画在同一个图上来检查, 如下面的图

对于给定的一个链, 如果其已经达到收敛, 那么任何感兴趣的量 可以通过计算样本的样本均值和样本方差进行推断. 从而,  $k$ 个链就有  $k$ 个可能的推断结

---

<sup>1</sup>A. Gelman and D.B. Rubin. A single sequence from the Gibbs sampler gives a false sense of security. In J. M. Bernardo, J. O. Berger, O.P. Dawid, and A.F.M. Smith, editors, Bayesian Statistics 4, Pages 625-631. Oxford University Press, Oxford, 1992



果. 那么如果链已经收敛, 这些推断就应该比较类似. 因此Gelman和Rubin提出使用ANOVA的方法进行分析.

假设有 $k$ 个链, 每个链有 $n$ 个样本, 感兴趣的量为 $\phi$ , 其在目标分布下有期望 $\mu$ 和方差 $\sigma^2$ . 记 $\phi_{jt}$ 表示链 $j$ 的第 $t$ 个样本时 $\phi$ 的值, 那么在混合样本中,  $\mu$ 的一个无偏估计为  $\hat{\mu} = \bar{\phi}_{..}$ . 而链之间的方差 $B/n$ 和链内的方差 $W$ 分别为

$$B/n = \frac{1}{k-1} \sum_{j=1}^k (\bar{\phi}_{j\cdot} - \bar{\phi}_{..})^2$$
$$W = \frac{1}{k(n-1)} \sum_{j=1}^k \sum_{t=1}^n (\phi_{jt} - \bar{\phi}_{j\cdot})^2$$

从而我们可以使用 $B$ 和 $W$ 加权进行估计 $\sigma^2$ :

$$\hat{\sigma}_+^2 = \frac{n-1}{n} W + \frac{B}{n}$$

如果初始值是从目标分布中抽取的,  $\hat{\sigma}_+^2$  就是 $\sigma^2$ 的无偏估计. 但是如果 初始值过度分散, 则就会高估 $\sigma^2$ .

考虑到估计量 $\hat{\mu}$ 的抽样波动性, 方差的估计取为  $\hat{V} = \hat{\sigma}_+^2 + B/(kn)$ .

比较混合和链内的推断可以通过

$$R = \frac{\hat{V}}{\sigma^2}$$

来进行. 称 $\sqrt{R}$ 为 *scale reduction factor*, *SRF*. 我们可以估计  $R$  为

$$\hat{R} = \frac{\hat{V}}{W}.$$

称 $\sqrt{\hat{R}}$ 为 *potential scale reduction factor*, *PSRF*. 当链达到收敛时 并且产生的数据很大时,  $\hat{R}$  应该趋于 1. Gelman & Rubin 建议的修正统计量为  $\sqrt{\hat{R} \frac{d}{d-2}}$ . 但此修正有误, Brooks and Gelman (1997)<sup>2</sup> 采用了一个修正的版本:

$$\hat{R}_c = \frac{d+3}{d+1} \hat{R}$$

---

<sup>2</sup> Brooks, SP. and Gelman, A. (1997) General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7, 434-455.

其中 $d$ 为 $\hat{V}$ 的自由度的估计. 这个修正是很微小的, 因为在收敛时,  $d$ 会很大.

**例 4 (Gelman-Rubin method)** 目标分布为 $N(0, 1)$ , 提议分布为 $N(X_t, \sigma^2)$ ,  $\phi_{jt}$  表示第 $j$ 个链前 $t$ 个样本的平均.

```
Gelman.Rubin <- function(psi) {  
  # psi[i,j] is the statistic psi(X[i,1:j])  
  # for chain in i-th row of X  
  psi <- as.matrix(psi)  
  n <- ncol(psi)  
  k <- nrow(psi)  
  psi.means <- rowMeans(psi)      #row means  
  B <- n * var(psi.means)        #between variance est.  
  psi.w <- apply(psi, 1, "var")  #within variances  
  W <- mean(psi.w)              #within est.  
  v.hat <- W*(n-1)/n + (B/(n*k)) #upper variance est.  
  r.hat <- v.hat / W            #G-R statistic  
  return(r.hat)  
}
```

[↑Code](#)

[↓Code](#)

下面的代码生成链:

```
normal.chain <- function(sigma, N, X1) {  
  #generates a Metropolis chain for Normal(0,1)  
  #with Normal(X[t], sigma) proposal distribution  
  #and starting value X1  
  x <- rep(0, N)  
  x[1] <- X1  
  u <- runif(N)  
  for (i in 2:N) {  
    xt <- x[i-1]  
    y <- rnorm(1, xt, sigma)      #candidate point  
    r1 <- dnorm(y, 0, 1) * dnorm(xt, y, sigma)  
    r2 <- dnorm(xt, 0, 1) * dnorm(y, xt, sigma)  
    r <- r1 / r2  
    if (u[i] <= r) x[i] <- y else  
      x[i] <- xt  
  }  
  return(x)  
}
```

[↑Code](#)



在下面的计算中, 方差 $\sigma^2$ 取的很小, 当提议分布的方差相比于目标分布的方法很小时, 链混合的就会很慢.

```
sigma <- .2      #parameter of proposal distribution
k <- 4          #number of chains to generate
n <- 15000      #length of chains
b <- 1000       #burn-in length

#choose overdispersed initial values
x0 <- c(-10, -5, 5, 10)

#generate the chains
X <- matrix(0, nrow=k, ncol=n)
for (i in 1:k)
\   X[i, ] <- normal.chain(sigma, n, x0[i])

#compute diagnostic statistics
psi <- t(apply(X, 1, cumsum))
```

```
for (i in 1:nrow(psi))
  psi[i,] <- psi[i,] / (1:ncol(psi))
print(Gelman.Rubin(psi))

#plot psi for the four chains
par(mfrow=c(2,2))
for (i in 1:k)
  plot(psi[i, (b+1):n], type="l",
       xlab=i, ylab=bquote(psi))
par(mfrow=c(1,1)) #restore default

#plot the sequence of R-hat statistics
rhat <- rep(0, n)
for (j in (b+1):n)
  rhat[j] <- Gelman.Rubin(psi[,1:j])
plot(rhat[(b+1):n], type="l", xlab="", ylab="R")
abline(h=1.1, lty=2)
```

[↓Code](#)

## 1.6 WinBUGS Introduction

BUGS(Bayesian inference Using Gibbs Sampling) 是使用MCMC算法进行Bayes计算的一个项目. WinBUGS 是该项目的一个软件产品. 目前WinBUGS的版本号为1.4.3.

WinBUGS所有的工作都是在一个*compound document*文档, 扩展名为`.odc`, 中进行. 该文档包含了模型程序代码, 数据, 以及可以将分析结果(图形,数据等)也保存在内. 下面我们来说明如何建立这样一个文档. 在WinBUGS里新建一个空白文档后(菜单File-New), 一个贝叶斯分析就从[建立模型](#), [数据](#), [初始值](#)三部分开始.

### 1.6.1 Building Bayesian models in WinBUGS

建立的一个Bayes模型, 包括: 指定似然和先验; 读入数据; 初始值. WinBUGS里模型里的参数/变量有三类:

1. 常量, 取固定值的量.

2. 随机部分, 通过一个概率分布来描述, 模型里的参数和响应变量都是随机变量, 分别通过先验分布和似然来描述. 随机部分 在进行MCMC算法时还要指定初始值.

3. 逻辑部分, 即通过一个数学表达式来刻画变量之间的关系.

模型的语法结构如下

```
model{  
variable~distribution(parameter1, parameter2,...)  
parameter1<-a+x*b  
a~distribution(a0,b0)  
b~distribution(a1,b1)  
parameter2~distribution(c,d)  
.....  
}
```

[↑Code](#)

[↓Code](#)

首先我们来介绍一下WinBUGS里 常用的数学函数, 和分布函数, 数学函

数作用在指定对象上后可以使用<-赋值到另外一个对象.

函数	意义
abs(x)	$ x $
cloglog(x)	$\log(-\log(1-x))$
cos(x)	$\cos(x)$
exp(x)	$\exp(x)$
equals(x1,x2)	$f(x_1, x_2) = 1, x_1 = x_2; 0, \text{otherwise}$
sin(x)	$\sin(x)$
inprod(x1[],x2[])	$\sum_{i=1}^n x_1[i]x_2[i]$
inverse(A[,])	$A^{-1}$
interp.lin(x,v1[],v2[])	$v_{2i} + \frac{x-v_{1i}}{v_{1,i+1}-v_{1i}}(v_{2,i+1} - v_{2i})$
logdet(A[,])	$\log A $
logfact(k)	$\log(k!)$
loggam(x)	$\log\Gamma(x)$
logit(x)	$\log\frac{x}{1-x}$
max(x1,x2)	$\max(x_1, x_2)$
min(x1,x2)	$\min(x_1, x_2)$
mean(x[])	$\frac{1}{n} \sum_{i=1}^n x_i$
sd(x[])	$\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 / (n-1)}$
phi(x)	$P(X \leq x), X \sim N(0, 1)$
pow(x,z)	$x^z$
sqr(x)	$\sqrt{x}$
sum(x[])	$\sum_{i=1}^n x_i$
rank(v[],k)	$\sum_{i=1}^n I(v_i \leq v_k)$
ranked(v[],k)	$v_s : \sum_{i=1}^n I(v_i \leq v_s) = k$
cut(x)	posterior of x is not updated by the likelihood
round(x)	round x to the closest integer
step(x)	$f(x) = 1, x \geq 0; 0, \text{otherwise}$
trunc(x)	truncation to the closest smaller than x integer

分布函数可以参看WinBUGS菜单*help-user manual*, 然后点击打开的文档中的distribution连接查看. 另外, **for**循环也可以使用, 其和R里的用法相同.

## 1.6.2 Model specification in WinBUGS

模型的指定包括似然和先验两部分.

**似然函数指定**, 即指定相应变量的分布

$$y \sim \text{distribution}(\vartheta)$$

其中 $\vartheta$ 和一些解释变量有关系.

$$\vartheta = h(\theta, x_1, \dots, x_p)$$

因此似然函数为

$$f(y|\theta) = \prod_{i=1}^n f(y_i|\vartheta_i = h(\theta, x_{i1}, \dots, x_{pi}))$$

此似然函数在WinBUGS里的代码如下

```
model{
  for(i in 1:n){
    y[i]~distribution.name(parameter1[i],parameter2[i],...)
    parameter1[i]<-function of theta and X's
    parameter2[i]<-function of theta and X's
    ...}
}
```

[↑Code](#)

[↓Code](#)

**先验指定** 指定完似然后,紧接着需要指定参数的先验分布:

```
theta1~distribution.name
theta2~distribution.name
...
```

[↑Code](#)

[↓Code](#)

### 1.6.3 Data and initial value specification

在WinBUGS里, 各种类型的数据是放在一个`list`里, 数据包括模型里 常变量的值, 如变量个数, 样本量等, 以及样本, MCMC算法的初始值等. 数据的读入可以通过两种方式: `rectangular`和`list`格式.

#### WinBUGS中的数值表示

##### 1. 向量

1.  $v[]$ , 向量 $v$ 的所有值.
2.  $v[i]$ , 向量 $v$ 的第 $i$ 个元素.
3.  $v[a : b]$ , 向量 $v$ 的第 $a$ 到第 $b$ 个元素.

向量可以在`list`函数里使用函数`c`来创建.

##### 2. 矩阵



1.  $M[,]$ , 矩阵 $M$ 的所有元素
2.  $M[i, j]$ , 矩阵 $M$ 的 $ij$ 元素
3.  $M[i, ]$ , 矩阵 $M$ 的第 $i$ 行所有元素
4.  $M[, j]$ , 矩阵 $M$ 的第 $j$ 列所有元素
5.  $M[n : m, k : l]$ , 矩阵 $M$ 的第 $n$ 至第 $m$ 行, 第 $k$ 至第 $l$ 列所有元素

二维矩阵可以推广到多维数组. 对于矩阵元素的访问和运算和R中的运算基本相同, 但是在R中, 对整个矩阵操作只需使用其名称, 而在WinBUGS中, 需要使用名称[,]的形式. 创建矩阵的命令也不同, 在WinBUGS中, 创建一个矩阵的语法格式为

```
matrix.name=structure(  
    .Data=c(value1,value2,...,valuek),  
    .Dim=c(row.number,col.number)  
)
```

[↑Code](#)

---

[↓Code](#)

矩阵`matrix.name`是由`.Data`中的元素在`.Dim`规定下按照**行顺序**生成的. 在上述语法中, `.Dim`如果是三维以上的, 则就会创建多维数值. 因此, 在`R/Plus`中, 使用

---

[↑Code](#)

```
> a<-structure(.Data=c(1:12),.Dim=c(3,4))#列顺序生成
> a
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

---

[↓Code](#)

而在WinBUGS中, 是按行顺序生成:

---

[↑Code](#)

```
> a
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

```
[1,]    1     2     3     4
[2,]    5     6     7     8
[3,]    9    10    11    12
```

[↓Code](#)

高维的数组也有这样的区别.

*rectangular* 格式

这种格式是使用数据的自然矩形形式. 其第一行为各变量的名称, 各变量的值以行排列的方式 排序, 最后一行以**END**结束, 并保持其后至少一个空行(1.4版本). 如

```
v1[]    v2[]    v3[]    v4[]    v5[]
val11  val112  val113  val14   val15
val21  val122  val123  val24   val25
val31  val132  val133  val34   val35
val41  val142  val143  val44   val45
END
```

#空行

*list*格式, 类似于R里的list格式, 语法格式为

```
list(variable1=value, variable2=value,...)
```

其中取值可以是向量, 矩阵或者数组.

**例 5 一个简单的数据指定例子** 假如我们的数据如下

y	x1	x2	gender	age
12	2	0.3	1	20
23	5	0.2	2	21
54	9	0.9	1	23
32	11	2.1	2	20

我们需要加入两个变量: 样本量( $n = 4$ )和变量个数( $p = 5$ ), 于是可以

```
list(n=4,p=5,y=c(12,23,54,32),x1=c(2,5,9,11),x2=c(0.3,0.2,0.9,2.1))
```

```
,gender=c(1,2,1,2),age=c(20,21,24,20))
```

如果需要使用矩阵形式, 则

```
list(n=4,p=5,datamatrix=structure(  
    .Data=c(12,2,0.3,1,20,23,5,0.2,2,  
            21,54,9,0.9,1,23,32,11,2.1,2,20),  
    .Dim=c(4,5)))
```

在WinBUGS里, 当模型完成编译后, 可以使用菜单*Info*里的*Node Info*来查看指定的对象.

使用*matrix*或者*array*还是不太方便, 好处是创建后数据格式后, 可以在编译模型时 一次读入所有数据. 但是我们也可以使用下面这个方式, 分多次读入需要的数据:

```
y[] x1[] x2[] gender[] age[]  
12   2  0.3   1     20
```

```
23    5  0.2    2      21
54    9  0.9    1      23
32   11  2.1    2      20
END
```

#空白行

这样可以使用 *model specification tool* 的 *load* 按钮多次读入需要的数据, 但是对于模型中的一些常数, 比如样本量, 变量个数等, 需要再创建一个 *list* 对象来读入.

初始值用于初始化MCMC算法. 它们的格式和上述 *list* 格式相同.

**例 6 一个完整的例子** 假设我们有10个数据, 来自于总体  $N(\mu, \sigma^2)$ , 我们想推断  $\mu, \sigma^2$ . 先验分布为  $\mu \sim N(0, 100), \sigma^2 \sim IG(0.01, 0.01)$ . 则整个代码如下

```
model{
  #likelihood
  for(i in 1:n)
```

[↑Code](#)

```
y[i]~dnorm(mu,tau)
#prior
mu~dnorm(0,0.01)
tau~dgamma(0.01,0.01)
#deterministic definition of variance
sigma.squared<-1/tau
sigma<-sqrt(sigma.squared)
}
```

DATA

```
list(n=10,y=c(1.806, 2.04, 1.423, -2.814, -1.196,
             -0.177, -0.233, -3.065, 0.871, 1.033))
```

INITIALS

```
list(mu=0,tau=1)
```

[↓ Code](#)

---

这里需要注意的是在WinBUGS里, 正态分布 $N(\mu, \sigma^2)$ 中标准差的指定是通过其逆来指定. 另外, 在一个代码很长的文档中, 我们可以使用菜单`tools`里的`creat fold`命令来隐藏 指定部分的代码, 以节约屏幕空间.

## 1.6.4 Compiling model and simulating values

在模型和数据代码完成后, 我们需要对模型进行编译, 才能开始进行MCMC模拟. 其步骤可以列为下面几步:

1. 打开 *model specification tool*
2. 检查模型的语法正确性.
3. 读入数据.
4. 编译模型.
5. 设置MCMC链的个数和初始值.
6. 运行MCMC算法, 产生随机数.
7. 使用 *Inference* 菜单里的 *samples* 来监视我们需要的参数.
8. 进行后续推断和分析.

**例 7 检测煤矿灾难事件数据的变点** Poisson分布(过程)常被用来对某个事件发生次数进行建模. 考虑英国煤矿1851年3月15日至1962年3月22日之间10多个煤矿191次爆炸事故数据. 从下面的图可以看出, 在某一年后, 灾难事



件数明显减少, 我们的目的就是估计这个变化点.

```
library(boot)      #for coal data
data(coal)
year <- floor(coal)
y <- table(year)
plot(y) #a time plot
```

[↑Code](#)

[↓Code](#)

首先我们从数据集 *coal* 中导出每年的爆炸次数数据:

```
y <- floor(coal[[1]])
y <- tabulate(y)
y <- y[1851:length(y)]
```

[↑Code](#)

[↓Code](#)

此即我们的观测数据. 对此计数数据, 考虑如下的模型

$$y_i \sim \text{Poisson}(\mu), i = 1, \dots, k;$$

$$y_j \sim \text{Poisson}(\lambda), j = i + 1, \dots, n.$$

即假设在某个年份(变点)之前, 灾难数服从一个Poisson分布, 而在此年后, 灾难数服从另一个Poisson分布. 对速率 $\mu$ 和 $\lambda$ 使用 $\log$ 函数表示. 因此, 在WinBUGS里的代码如下

```
model {  
  for( i in 1 : n ) {  
    y[i] ~ dpois(mu[i])  
    log(mu[i]) <- b[1] + step(i - k) * b[2]  
  }  
  for (j in 1:2) {  
    b[j] ~ dnorm( 0.0,1.0E-6)  
  }  
  k ~ dunif(1,n)  
}
```

[↑Code](#)[↓Code](#)

数据和初始值指定, 可以通过R里的函数**dput**来得到和WinBUGS里格式最

相近的一个形式, 简单修改就可以得到

#### DATA

```
list(n=112,y=c(4, 5, 4, 1, 0, 4, 3, 4, 0, 6, 3, 3, 4, 0, 2, 6, 3, 3, 5, 4,  
5, 3, 1, 4, 4, 1, 5, 5, 3, 4, 2, 5, 2, 2, 3, 4, 2, 1, 3, 2, 2,  
1, 1, 1, 1, 3, 0, 0, 1, 0, 1, 1, 0, 0, 3, 1, 0, 3, 2, 2, 0, 1,  
1, 1, 0, 1, 0, 1, 0, 0, 0, 2, 1, 0, 0, 0, 1, 1, 0, 2, 3, 3, 1,  
1, 2, 1, 1, 1, 1, 2, 3, 3, 0, 0, 0, 1, 4, 0, 0, 0, 1, 0, 0, 0,  
0, 0, 1, 0, 0, 1, 0, 1))
```

[↑Code](#)

#### INITIALS

```
list(b=c(0,0),k=50)
```

[↓Code](#)

然后就可以进行MCMC模拟了. 过程见课堂展示.

下面我们介绍另外一种方式. 使用R里的包**R2WinBUGS**来进行. 安装好WinBUGS 和此包后, 就可以在R里使用WinBUGS进行贝叶斯分析了. 在R里使用WinBUGS进行贝叶斯分析的主要函数为**bugs**, 其语法如下

[↑Code](#)

```
> model.sim <- bugs(data, inits, parameters, "model.bug")
```

[↓Code](#)

则WinBUGS就会在后台(默认)运行. 对上面的这个例子, 一种做法是首先将模型的代码存入一个文本文档, 比如 "coal.bug", 在R里执行的代码如下

```
> n=112
> y=c(4,5,4,1,0,4,3,4,0,6,
+ 3,3,4,0,2,6,3,3,5,4,5,3,1,4,4,1,5,5,3,4,2,5,2,2,3,4,2,1,3,2,
+ 1,1,1,1,1,3,0,0,1,0,1,1,0,0,3,1,0,3,2,2,
+ 0,1,1,1,0,1,0,1,0,0,0,2,1,0,0,0,1,1,0,2,
+ 2,3,1,1,2,1,1,1,1,2,4,2,0,0,0,1,4,0,0,0,
+ 1,0,0,0,0,0,1,0,0,1,0,0)
> data=list("n","y")
> parameters <- c("k","b")
> inits = function() {list(b=c(0,0),k=50)}
> coal.sim <- bugs (data, inits, parameters,
+ "coal.bug", n.chains=3, n.iter=10000, ,bugs.directory="D:/WinBUGS14")
> attach.bugs(coal.sim)
> print(coal.sim)
```

[↑Code](#)

```
> par(mfrow=c(2,1))
> plot(density(b[,1]),xlab="beta1")
> plot(density(b[,2]),xlab="beta2")
```

[↓Code](#)

另外一种做法是使用R2WinBUGS包里的[write.model](#)函数, 直接在R script里将模型存在临时目录里. 然后就可以 使用bugs函数来读取了.

```
coal<-function(){
  for( i in 1 : n ) {
    y[i] ~ dpois(mu[i])
    log(mu[i]) <- b[1] + step(i - k) * b[2]
  }
  for (j in 1:2) {
    b[j] ~ dnorm( 0.0,1.0E-6)
  }
  k ~ dunif(1,n)
}
write.model(coal,"coal.bug")
coal.sim <- bugs (data, inits, parameters,
```

[↑Code](#)

```
"coal.bug", n.chains=3, n.iter=10000, bugs.directory="D:/WinBUGS14")
```

[↓Code](#)

---

在R里还可以调用包**`coda`**来作更多的分析. 如绘制自相关图, 收敛诊断等等. 请参考**`coda`**的说明文档.