MCMC 方法初步

王宁宁

Tuesday, May 26, 2015

假设$X \sim \pi(x)$，MC 算法是利用分布π的随机数来计算 E(f(X))，而 MCMC 是构造一个稳定分布是π的马氏链来估计 E(f(X))

$$\mathbb{E}(f(X)) \approx \frac{1}{N}\sum_{j=1}^{N} f(X_j),$$

MCMC的实现算法有几十种之多，具体可以看 这里 。

下面考虑一个简单的使用 Metropolis–Hastings 算法的例子：使用 MCMC 估计古典一元回归模型（三个参数）：

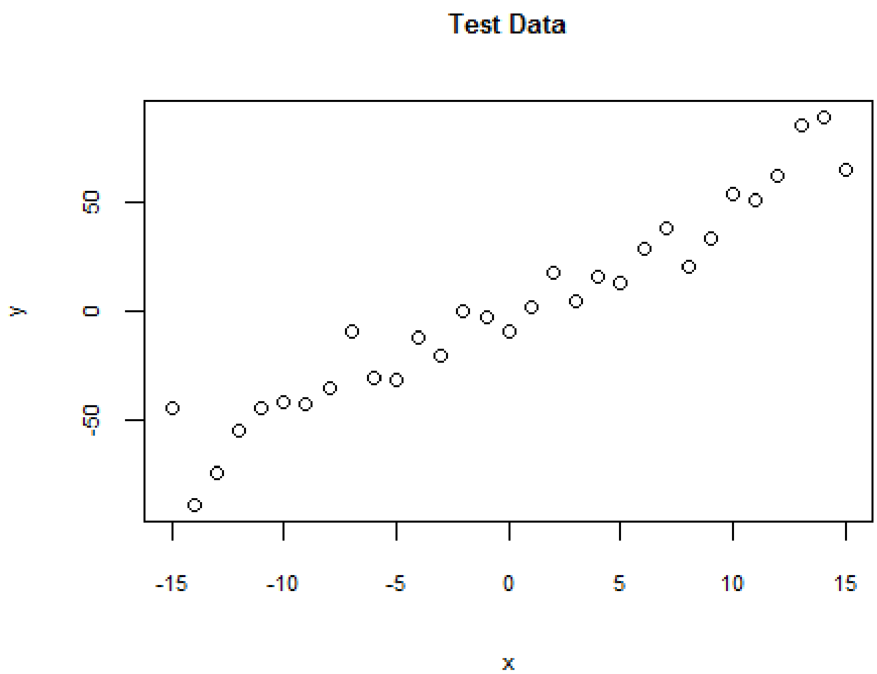
Y=aX+b

设定模型的真值：

```
trueA <- 5
trueB <- 0
trueSd <- 10
sampleSize <- 31
set.seed(6111)
```

模拟数据：

```
# create independent x-values
x <- (-(sampleSize-1)/2):((sampleSize-1)/2)
# create dependent values according to ax + b + N(0,sd)
y <-  trueA * x + trueB + rnorm(n=sampleSize,mean=0,sd=trueSd)
win.graph(width=9.875, height=7.5,pointsize=8)
plot(x,y, main="Test Data")
```



模型有三个参数：a，b，sd

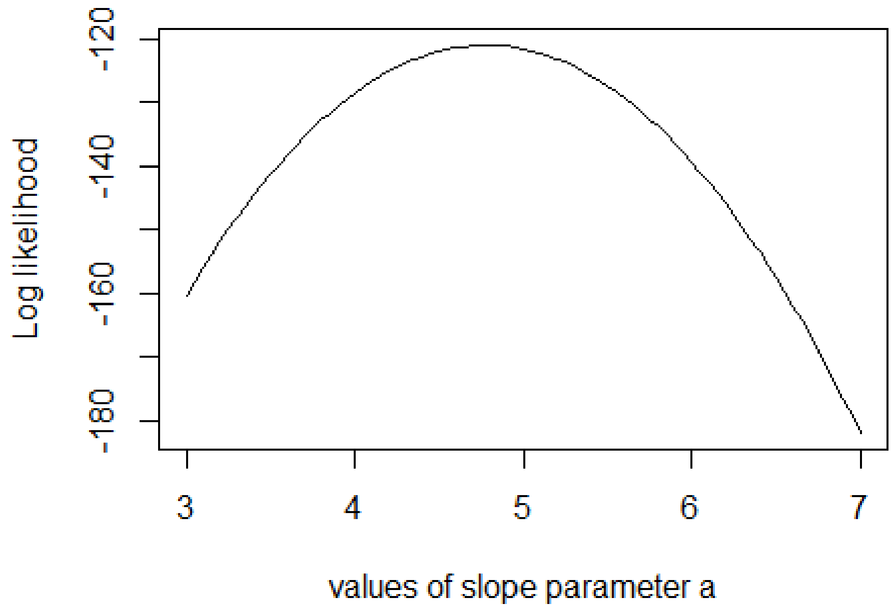
对数似然函数：

```
likelihood <- function(param){
  a = param[1]
  b = param[2]
  sd = param[3]

  pred = a*x + b
  singlelikelihoods = dnorm(y, mean = pred, sd = sd, log = T)
  sumll = sum(singlelikelihoods)
  return(sumll)
}
```

似然函数的图形：（为简便起见，只考虑第一个参数的似然函数，其余两个参数代入真值。）

```
# plot the likelihood profile of the slope a
slopevalues <- function(x){return(likelihood(c(x, trueB, trueSd)))}
slopelikelihoods <- lapply(seq(3, 7, by=.05), slopevalues )
plot (seq(3, 7, by=.05), slopelikelihoods , type="l", xlab = "values of
 slope parameter a", ylab = "Log likelihood")
```



利用贝叶斯分析估计参数，首先要设定先验：

a：均匀分布 U(0,10) b: 正态分布 N(0,5^2) sd：均匀分布 U(0,30)

```
prior <- function(param){
  a = param[1]
  b = param[2]
  sd = param[3]
  aprior = dunif(a, min=0, max=10, log = T)
  bprior = dnorm(b, sd = 5, log = T)
  sdprior = dunif(sd, min=0, max=30, log = T)
  return(aprior+bprior+sdprior)
}
```

后验（联合）分布：

```
posterior <- function(param){
  return (likelihood(param) + prior(param))
}
```

思考：为什么后验分布是这种形式？

Metropolis 算法：

Algorithm 1 Metropolis-Hastings algorithm

Initialize $x^{(0)} \sim q(x)$
for iteration $i = 1, 2, \ldots$ **do**
Propose: $x^{cand} \sim q(x^{(i)}|x^{(i-1)})$
Acceptance Probability:
$$\alpha(x^{cand}|x^{(i-1)}) = \min\left\{1, \frac{q(x^{(i-1)}|x^{cand})\pi(x^{cand})}{q(x^{cand}|x^{(i-1)})\pi(x^{(i-1)})}\right\}$$
$u \sim \text{Uniform}\ (u; 0, 1)$
if $u < \alpha$ **then**
Accept the proposal: $x^{(i)} \leftarrow x^{cand}$
else
Reject the proposal: $x^{(i)} \leftarrow x^{(i-1)}$
end if
end for

更深入的关于metropolis 算法的介绍可以看这里(Metropolis–Hastings algorithm)。

```
######## Metropolis algorithm ################

proposalfunction <- function(param){
  return(rnorm(3,mean = param, sd= c(0.1,0.5,0.3)))
}#这个是筛选函数

run_metropolis_MCMC <- function(startvalue, iterations){
  chain = array(dim = c(iterations+1,3))
  chain[1,] = startvalue
  for (i in 1:iterations){
    proposal = proposalfunction(chain[i,])
    probab = exp(posterior(proposal) - posterior(chain[i,]))#接收概率
    if (runif(1) < probab){
      chain[i+1,] = proposal
    }else{
      chain[i+1,] = chain[i,]
    }
  }
  return(chain)
}
```

我们这里的筛选接收概率是：prob(new)/prob(old)=exp(log(prob(new))-log(prob(old)))

Metropolis–Hastings 算法的关键好处：**取舍函数中，只需要用到后验密度的核。**

运行，设定初始值是（3，3，20），运行 10000 次，除去前 5000 次。

```
startvalue = c(3,3,20)
chain = run_metropolis_MCMC(startvalue, 10000)
burnIn = 5000
```

```r
#acceptance = 1-mean(duplicated(chain[-(1:burnIn),]))
#acceptance
```

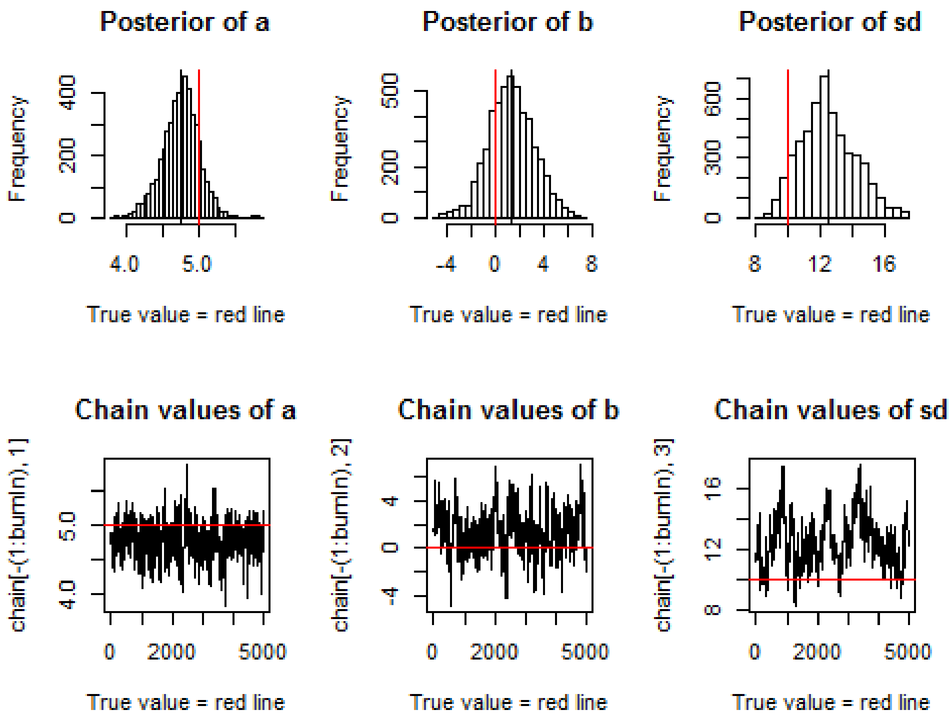
三个参数贝叶斯估计值:

```r
summary(chain[-(1:burnIn),])
```

```
##        V1              V2                V3
##  Min.   :3.83   Min.   :-4.949   Min.   : 8.28
##  1st Qu.:4.61   1st Qu.: 0.025   1st Qu.:11.30
##  Median :4.77   Median : 1.264   Median :12.31
##  Mean   :4.76   Mean   : 1.310   Mean   :12.48
##  3rd Qu.:4.92   3rd Qu.: 2.569   3rd Qu.:13.59
##  Max.   :5.87   Max.   : 7.013   Max.   :17.49
```

结果展示:

```r
par(mfrow = c(2,3))
hist(chain[-(1:burnIn),1],nclass=30, , main="Posterior of a", xlab="Tru
e value = red line" )
abline(v = mean(chain[-(1:burnIn),1]))
abline(v = trueA, col="red" )
hist(chain[-(1:burnIn),2],nclass=30, main="Posterior of b", xlab="True
value = red line")
abline(v = mean(chain[-(1:burnIn),2]))
abline(v = trueB, col="red" )
hist(chain[-(1:burnIn),3],nclass=30, main="Posterior of sd", xlab="True
 value = red line")
abline(v = mean(chain[-(1:burnIn),3]) )
abline(v = trueSd, col="red" )
plot(chain[-(1:burnIn),1], type = "l", xlab="True value = red line" , m
ain = "Chain values of a", )
abline(h = trueA, col="red" )
plot(chain[-(1:burnIn),2], type = "l", xlab="True value = red line" , m
ain = "Chain values of b", )
abline(h = trueB, col="red" )
plot(chain[-(1:burnIn),3], type = "l", xlab="True value = red line" , m
ain = "Chain values of sd", )
abline(h = trueSd, col="red" )
```



与 OLS 对比：

```
summary(lm(y~x))

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.061  -8.413  -0.537   5.624  25.876
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.112      2.154    0.52     0.61
## x              4.781      0.241   19.85   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12 on 29 degrees of freedom
## Multiple R-squared:  0.931,  Adjusted R-squared:  0.929
## F-statistic:  394 on 1 and 29 DF,  p-value: <2e-16
```