STAT 428 Statistical Computing

Homework 5 Solutions

Department of Statistics, University of Illinois at Urbana-Champaign

Ex 1 (Rizzo 9.3)

It is worth noticing that the computation acceptance ratio could be simplified as:

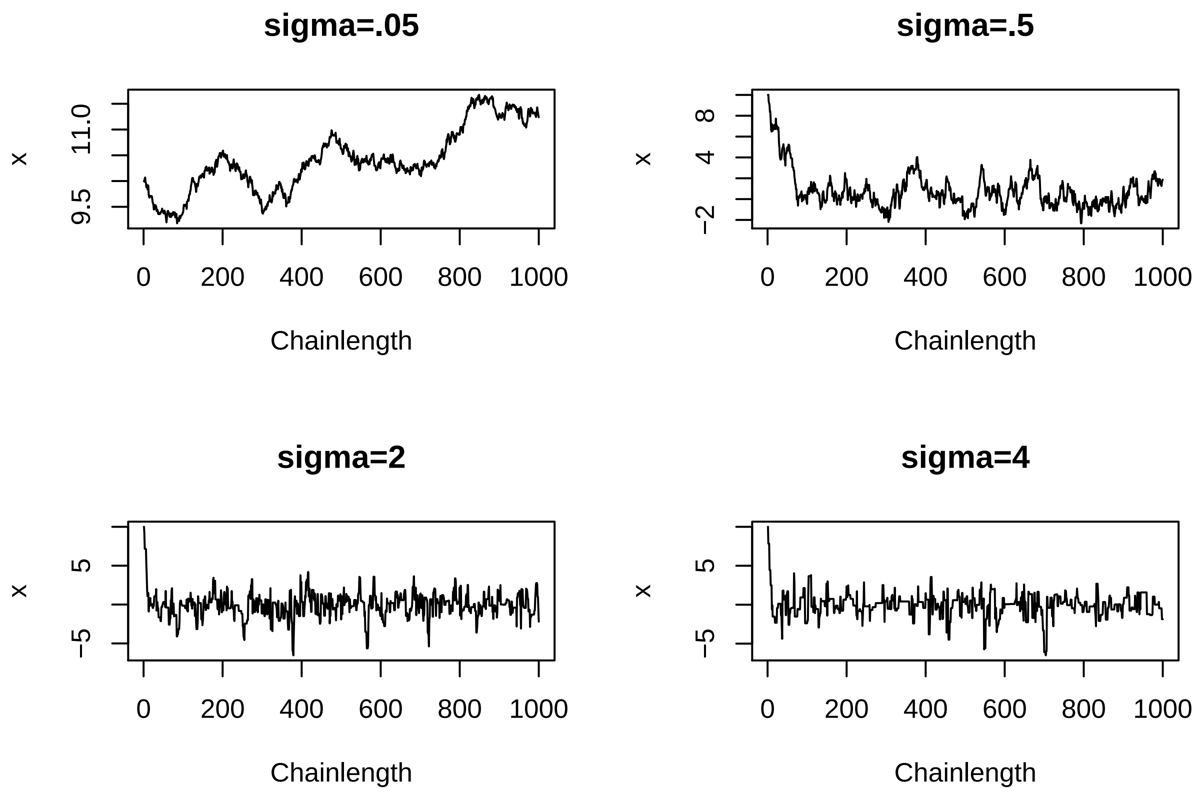
$$\frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)} = \frac{\frac{1}{\pi(1+y^2)} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(X_t-y)^2}{2\sigma^2}}}{\frac{1}{\pi(1+X_t^2)} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-X_t)^2}{2\sigma^2}}} = \frac{1+X_t^2}{1+y^2}$$

```r
m <- 10000
x <- numeric(m)
sigma = 1
#generate x0
x[1] <- rnorm(1,0,sigma)
k <- 0
u <- runif(m)
for (i in 2:m) {
  xt <- x[i-1]
  y <- rnorm(1, xt, sigma)

  #num <- dcauchy(y) * dnorm(xt, y, sigma)
  #den <- dcauchy(xt) * dnorm(y, xt, sigma)
  num <- (1+xt^2)
  den <- (1+y^2)
  if (u[i] <= num/den)
    x[i] <- y
  else {
    x[i] <- xt
    k <- k+1 #y is rejected
  }
}

dx = x[1001:10000]
df = rbind(quantile(dx,seq(.1,.9,.1)),qcauchy(seq(.1,.9,.1)))
rownames(df) = c('MH sampler','theoretical')
df
```

```
##                   10%       20%        30%        40%        50%       60%
## MH sampler  -2.645212 -1.323842 -0.7162606 -0.2949895 0.01213253 0.3299405
## theoretical -3.077684 -1.376382 -0.7265425 -0.3249197 0.00000000 0.3249197
##                   70%      80%      90%
## MH sampler  0.7162924 1.282647 2.376764
## theoretical 0.7265425 1.376382 3.077684
```

Ex 2 (Rizzo 9.4)

```r
dlaplace = function(x){
  return (1/2*exp(-abs(x)))
}

rw.Metropolis=function(sigma,x0,N){
  x=numeric(N)
  x[1]=x0
  u=runif(N)
  k=0 #number of accepted proposals
  for(i in 2:N){
    y=rnorm(1,x[i-1],sigma)
    ratio=dlaplace(y)/dlaplace(x[i-1])
    accept=u[i]<=ratio
    x[i]=y*accept+x[i-1]*(1-accept)
    k=k+accept
  }
  return(list(x=x,k=k))
}

N=1000
sigma=c(.05,.5,2,4)
x0=10 ## a huge outlier for a standard laplace dist
rw1=rw.Metropolis(sigma[1],x0,N)
rw2=rw.Metropolis(sigma[2],x0,N)
rw3=rw.Metropolis(sigma[3],x0,N)
rw4=rw.Metropolis(sigma[4],x0,N)

par(mfrow=c(2,2))
Chainlength=1:N
plot(Chainlength,rw1$x,ylab="x",type="l")
title("sigma=.05")
plot(Chainlength,rw2$x,ylab="x",type="l")
title("sigma=.5")
plot(Chainlength,rw3$x,ylab="x",type="l")
title("sigma=2")
plot(Chainlength,rw4$x,ylab="x",type="l")
title("sigma=4")
```



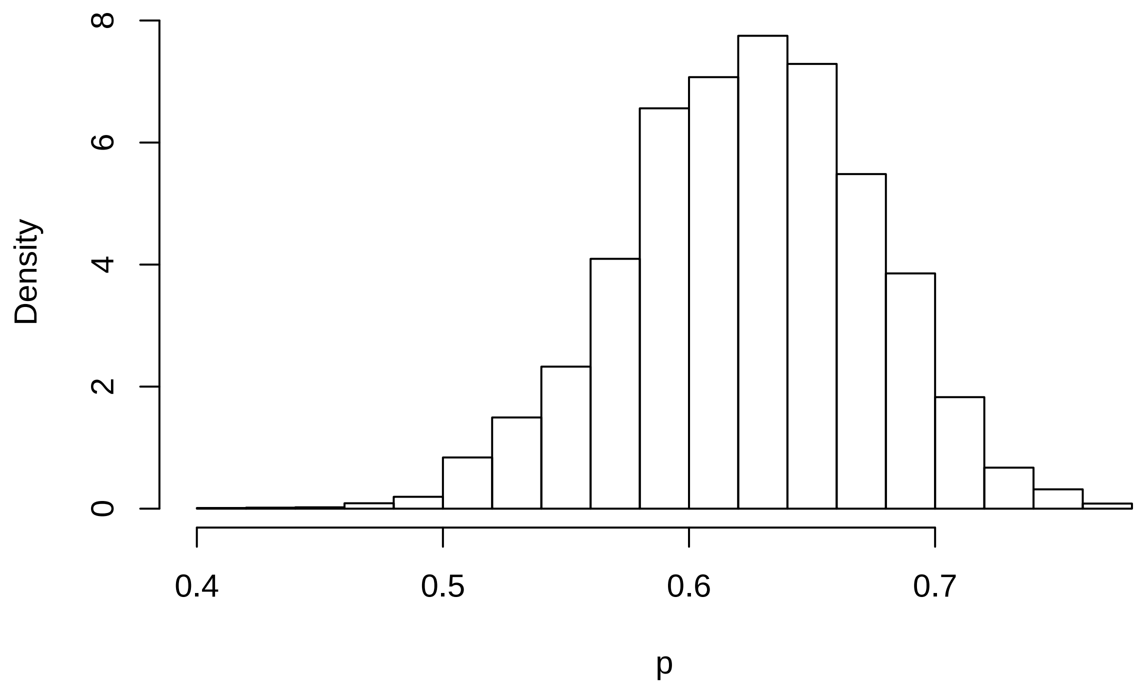
```r
print(c(rw1$k,rw2$k,rw3$k,rw4$k)/N)
```

```
## [1] 0.984 0.846 0.538 0.324
```

The chain with larger variance converges faster

Ex 3 (Rizzo 9.6)

```r
m = 10000
w = 0.25
categories = c(125,18,20,34)

prob <- function(theta, category) {
  if (theta < 0 || theta > 1) return (0)
  return( (1/2+theta/4)^category[1] * ((1-theta)/4)^category[2] *
    ((1-theta)/4)^category[3] * (theta/4)^category[4])
}
x = numeric(m)
u <- runif(m) #for accept/reject step
v <- runif(m,-w,w) #proposal distribution
x[1] <- runif(1,0,1)
for (i in 2:m) {
  y = x[i-1]+v[i]
  num = prob(y,categories)
  den = prob(x[i-1],categories)
  if (u[i] <= num/den)
    x[i] <- y
  else
    x[i] <- x[i-1]
}
hist(x[1001:m],main=" ",xlab="p",prob=TRUE)
```



```
mean(x[1001:m])
```

```
## [1] 0.6243238
```

```
sd(x[1001:m])
```

```
## [1] 0.04995817
```

Ex 4 (Rizzo 9.8)

```
N = 10000
X = matrix(0,N,2)
n = 20
a = 3
b = 5
X[1,] = c(5,.5)
for (i in 2:N) {
  X[i,1] <- rbinom(1, n, X[i-1,2])
  X[i,2] <- rbeta(1, X[i,1]+a, n-X[i,1]+b)
}
burn = 1000
apply(X[(burn+1):N,],2,mean)
```

```
## [1] 7.6147778 0.3814061
```

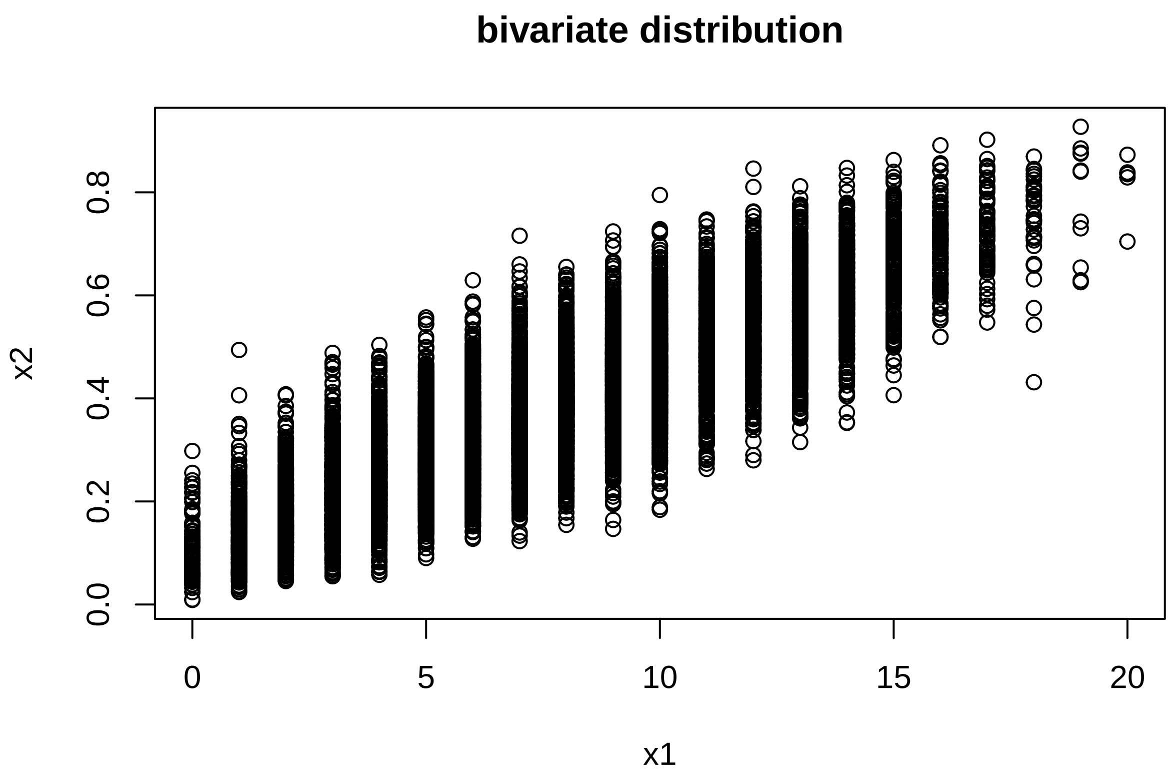
```
apply(X[(burn+1):N,],2,sd)
```

```
## [1] 3.8585018 0.1634085
```

```
cor(X[burn:N,1],X[burn:N,2])
```

```
## [1] 0.8451078
```

```
plot(X[(burn+1):N,1],X[(burn+1):N,2],main=" bivariate distribution ",xlab="x1", ylab="x2")
```



Ex 5 (Rizzo 9.11)

```
require('coda')
```

```
## Loading required package: coda
```

```
## Warning: package 'coda' was built under R version 3.3.2
```

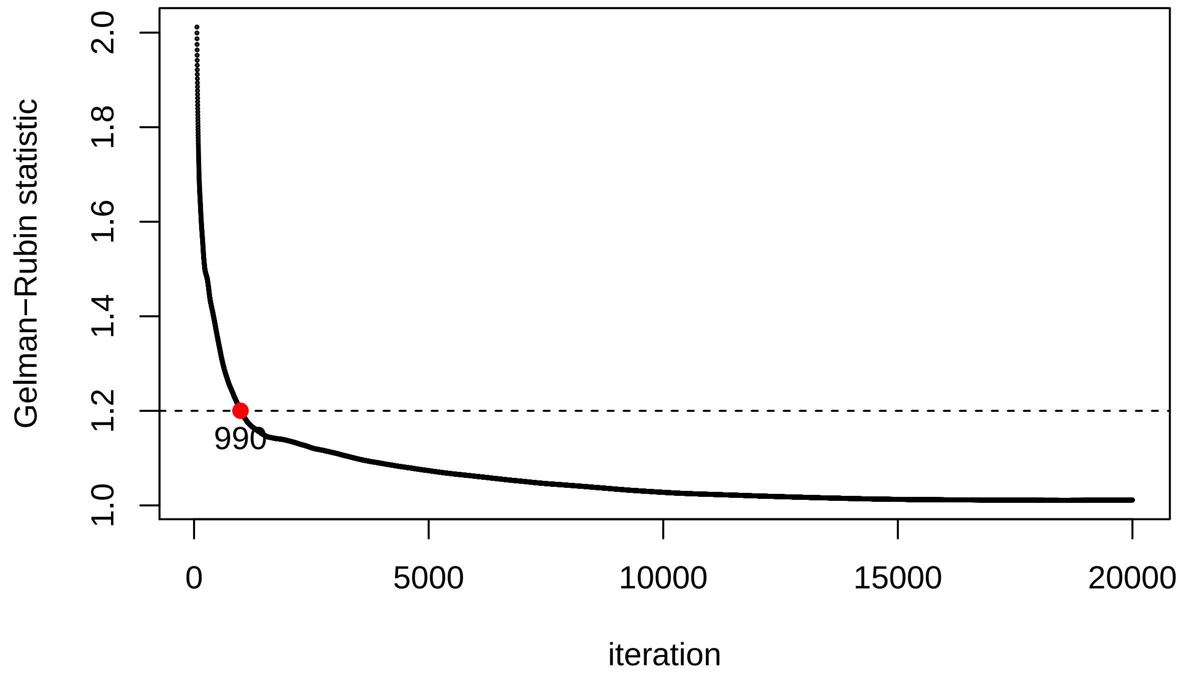
```
Gelman.Rubin=function(psi){
  psi=as.matrix(psi)
```

```r
  n=ncol(psi)
  k=nrow(psi)
  psi.means=rowMeans(psi) #row means
  B=n*var(psi.means) #between variance estimate
  psi.w=apply(psi,1,var) #within variance
  W=mean(psi.w) #within estimate
  v.hat=W*(n-1)/n+(B/n) # upper var est
  r.hat=v.hat/W # G-R statistic
  return(r.hat)
}

b <- .2 #actual value of beta
w <- .25 #width of the uniform support set
m <- 5000 #length of the chain
days <- 250
i <- sample(1:5, size=days, replace=TRUE,prob=c(1, 1-b, 1-2*b, 2*b, b))
win <- tabulate(i)
prob <- function(y, win) {
  # computes (without the constant) the target density
  if (y < 0 || y >= 0.5)
    return (0)
  return((1/3)^win[1] * ((1-y)/3)^win[2] * ((1-2*y)/3)^win[3] *((2*y)/3)^win[4] * (y/3)^win[5])
}

mc.chain=function(m,X1){
  u=runif(m)
  x=numeric(m)
  x[1] = X1
  v=runif(m,-w,w)
  for( i in 2:m){
    y=x[i-1]+v[i]
    ratio=prob(y,win)/prob(x[i-1],win)
    accept=u[i]<=ratio
    x[i]=y*accept+x[i-1]*(1-accept)
  }
  return(x)
}

k=5 # number of chains
n=20000 # length of chains
x0 = c(.01,.05,.1,.2,.4)
X=matrix(0,k,n)
for( j in 1:k)
  X[j,]=mc.chain(n,x0[j])

psi=t(apply(X,1,cumsum))
for(i in 1:nrow(psi))
  psi[i,]=psi[i,]/(1:ncol(psi))
print(Gelman.Rubin(psi))
```

```
## [1] 1.011446
```

```r
rhat=rep(0,n)
for(j in 1:n)
```

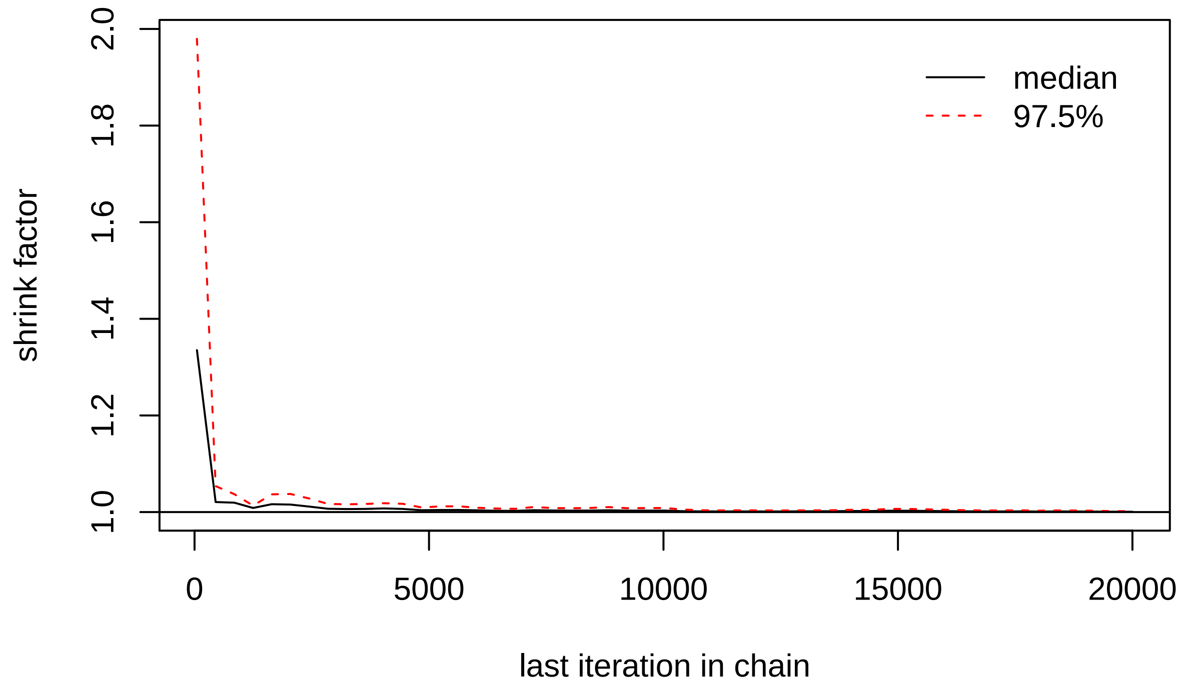
```
  rhat[j]=Gelman.Rubin(psi[,1:j])

start_idx = min(which(rhat[2:n]<2))
key_idx = min(which(rhat[2:n]<1.2))
plot(start_idx:n,rhat[start_idx:n],cex = .2,xlab = 'iteration', ylab='Gelman-Rubin statistic')
abline(h=1.2,lty=2)
points(key_idx, rhat[key_idx], col = "red", pch=19)
text(key_idx, rhat[key_idx],labels=key_idx,pos=1)
```



```
dm = mcmc.list(mcmc(X[1,]),mcmc(X[2,]),mcmc(X[3,]),mcmc(X[4,]),mcmc(X[5,]))
gelman.diag(dm)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1          1
```

```
gelman.plot(dm)
```


shrink factor
2.0
1.8
1.6
1.4
1.2
1.0
median
97.5%
0
5000
10000
15000
20000
last iteration in chain