MATH 598: TOPICS IN STATISTICS

TEMPERING AND ANNEALING IN MONTE CARLO

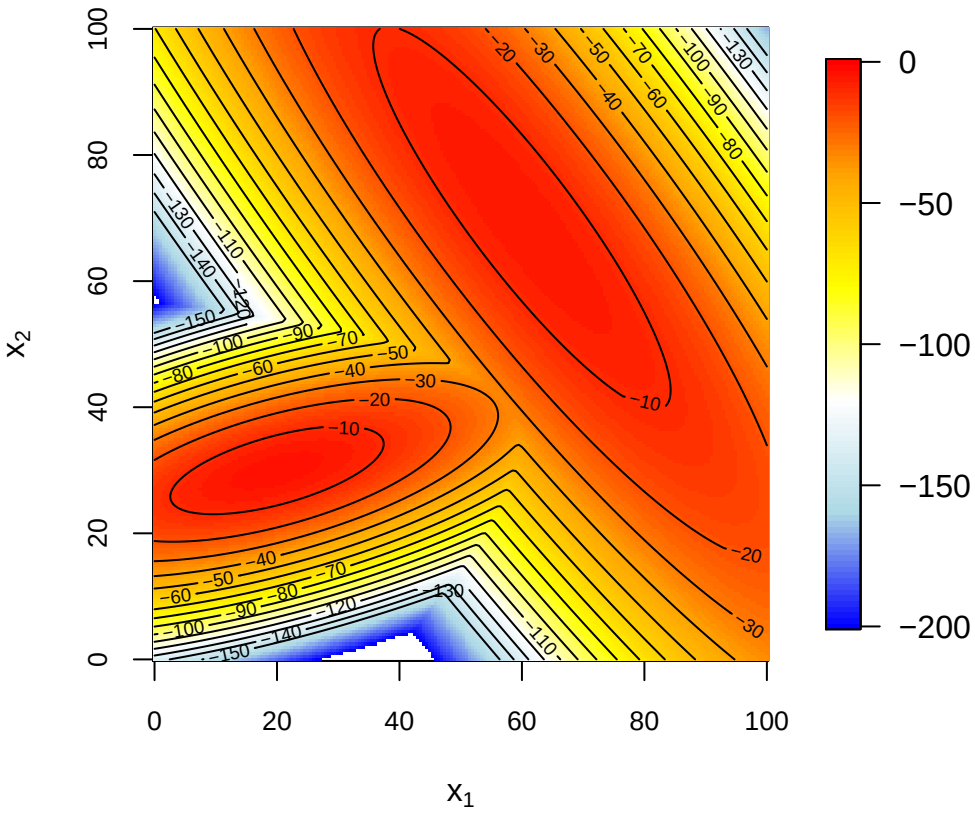
Suppose our target density is

$$\pi(\mathbf{x}) \equiv \omega Normal_2(\mu_1, \Sigma_1) + (1-\omega) Normal_2(\mu_2, \Sigma_2)$$

where μ_1 and μ_2 are relatively well separated. Ordinary Metropolis-Hastings methods can struggle so sample the pdf properly as they get stuck in local modes around one of the μ values. In the following example we have $\omega = 1/2$, and

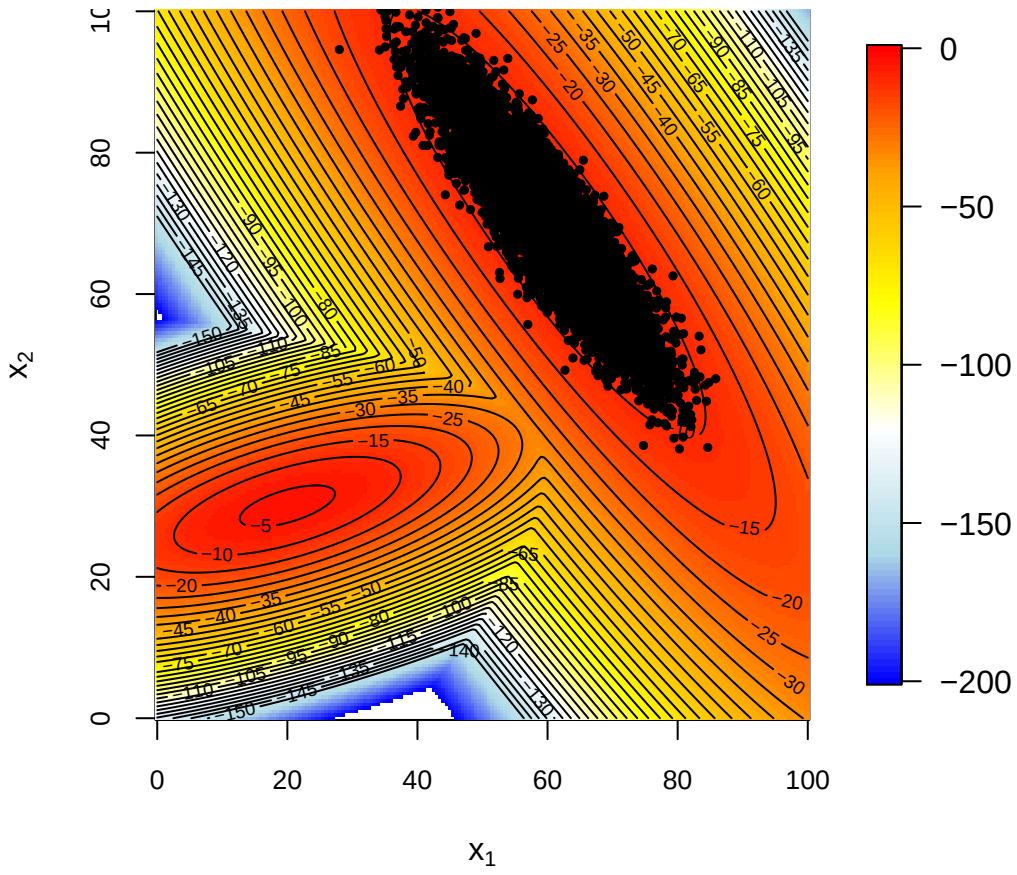
$$\mu_1 = (20, 30) \qquad \mu_2 = (60, 70) \qquad \Sigma_1 = \begin{bmatrix} 25 & 6 \\ 6 & 4 \end{bmatrix} \qquad \Sigma_2 = \begin{bmatrix} 64 & -72 \\ -72 & 100 \end{bmatrix}$$

```
set.seed(34)
xv<-yv<-seq(0,100,0.5)
require(mvnfast)
mu1<-c(20,30)
mu2<-c(60,70)

Sig1<-diag(c(5,2)) %*% matrix(c(1,0.6,0.6,1),2,2) %*% diag(c(5,2))
Sig1

+      [,1] [,2]
+ [1,]   25    6
+ [2,]    6    4

Tau1<-solve(Sig1)
Sig2<-diag(c(8,10)) %*% matrix(c(1,-0.9,-0.9,1),2,2) %*% diag(c(8,10))
Sig2

+      [,1] [,2]
+ [1,]   64  -72
+ [2,]  -72  100

Tau2<-solve(Sig2)

joint.pdf<-function(xv,yv,m1,m2,S1,S2,Tmp=1){
        return(log(dmvn(c(xv,yv),m1,S1)+dmvn(c(xv,yv),m2,S2))/Tmp)
}
f <- Vectorize(joint.pdf,vectorize.args=c("xv","yv"))
fmat<-outer(xv,yv,f,m1=mu1,m2=mu2,S1=Sig1,S2=Sig2)


library(fields,quietly=TRUE)
par(pty='s',mar=c(4,4,2,2))
colfunc <- colorRampPalette(c("blue","lightblue","white","yellow","orange","red"))
image.plot(xv,yv,fmat,col=colfunc(100),zlim=range(-200,-0),
           xlab=expression(x[1]),ylab=expression(x[2]),cex.axis=0.8)
contour(xv,yv,fmat,add=T,levels=seq(-150,0,by=10))
```

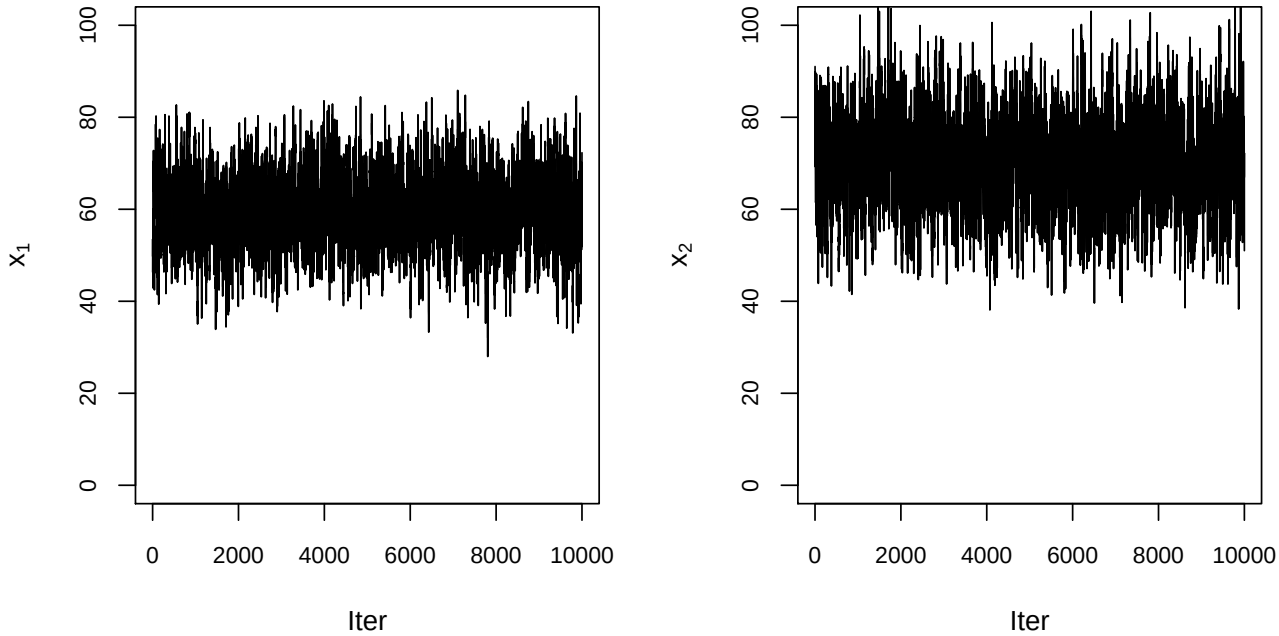


A Metropolis algorithm implementation illustrates the difficulty. The chain gets stuck in one mode, and cannot traverse the deep valley between the two modes.

```
old.x<-runif(2,0,100)
old.like<-log(dmvn(old.x,mu1,Sig1)+dmvn(old.x,mu2,Sig2))
nburn<-1000;nsamp<-10000;nthin<-10
nits<-nburn+nsamp*nthin
x.samp0<-array(0,c(nsamp,2))
like.val0<-matrix(0,nsamp)
ico<-0
for(iter in 1:nits){
        #Mutation
        new.x<-old.x+rmvn(1,c(0,0),diag(c(10,10)))
        new.like<-log(dmvn(new.x,mu1,Sig1)+dmvn(new.x,mu2,Sig2))
        if(log(runif(1)) < new.like-old.like){
                old.like<-new.like
                old.x<-new.x
        }
        if(iter > nburn & iter %% nthin == 0){
                ico<-ico+1
                x.samp0[ico,]<-old.x
                like.val0[ico,]<-old.like
        }
}
par(pty='s',mar=c(4,4,0,2))
image.plot(xv,yv,fmat,col=colfunc(100),zlim=range(-200,-0),
           xlab=expression(x[1]),ylab=expression(x[2]),cex.axis=0.8)
contour(xv,yv,fmat,add=T,levels=seq(-150,0,by=5));points(x.samp0,pch=19,cex=0.5)
```




```
par(mfrow=c(1,2),mar=c(4,4,1,2))
plot(x.samp0[,1],ylim=range(0,100),xlab='Iter',ylab=expression(x[1]),type='l',cex.axis=0.8)
plot(x.samp0[,2],ylim=range(0,100),xlab='Iter',ylab=expression(x[2]),type='l',cex.axis=0.8)
```



To overcome this we can use a population approach with tempered distributions. Let

$$\pi_m(x) \propto \{\pi(x)\}^{1/T_m} \qquad m = 1, 2, \ldots, M$$

where

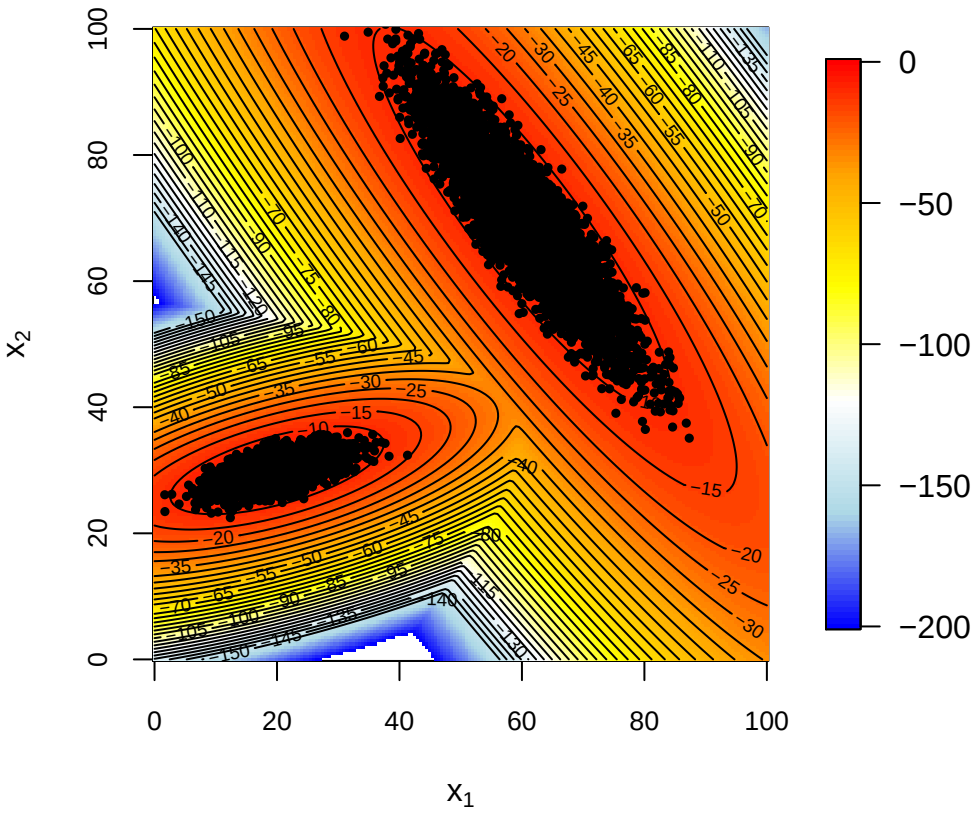
$$1 = T_1 < T_2 < \cdots < T_M$$

yield tempered forms of π. We run M parallel chains each with its own target, and then allow for exchange between the tempered chains and the target chain $m = 1$. In the following implementation, we use

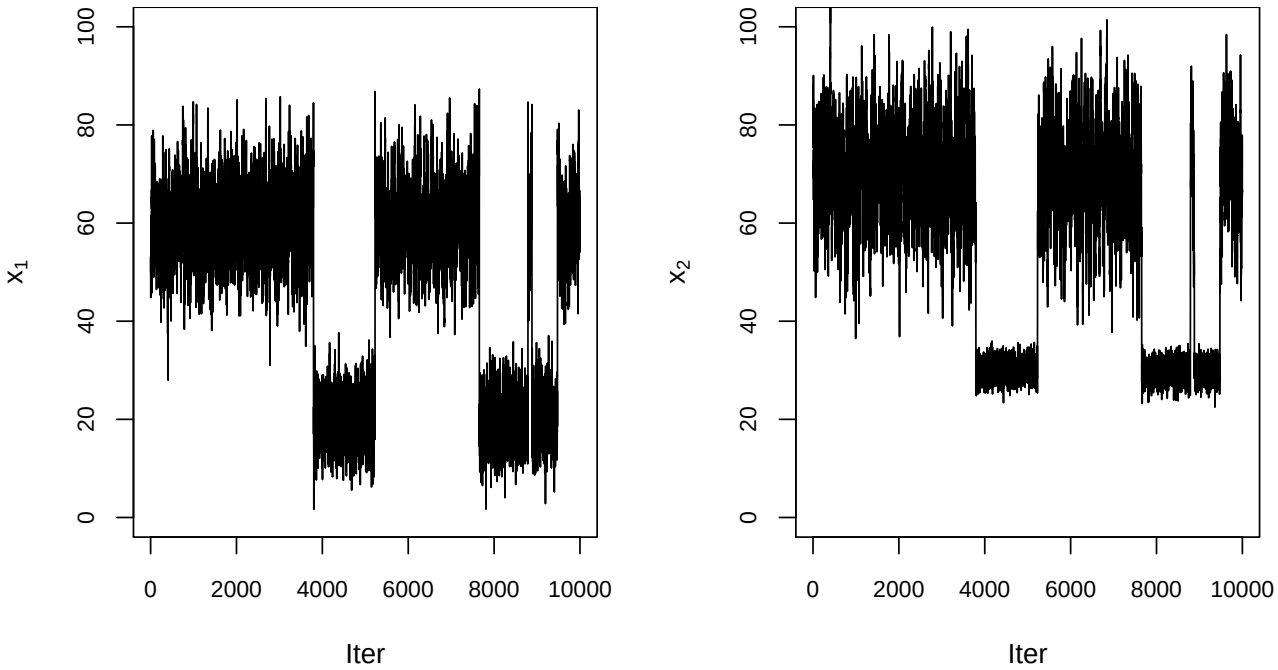
- **Mutation moves:** a standard Metropolis algorithm is used for each chain.
- **Exchange moves:** an exchange move between chain $m = 1$ and a randomly selected chain l say is proposed – this involves merely a swap of the current values within each chain, and an acceptance of the move with probability

$$\min\left\{1, \frac{\pi_l(x_m)\pi_m(x_l)}{\pi_l(x_l)\pi_m(x_m)}\right\}$$

```
M<-5
set.seed(3764)
Temps<-1+10*c(0:(M-1))/M

old.x<-matrix(runif(2*M,0,100),M,2)
old.like<-log(dmvn(old.x,mu1,Sig1)+dmvn(old.x,mu2,Sig2))/Temps

x.samp<-array(0,c(nsamp,M,2))
like.val<-matrix(0,nsamp,M)
ico<-0
for(iter in 1:nits){
        #Mutation
        new.x<-old.x+rmvn(M,c(0,0),diag(c(10,10)))
        new.like<-log(dmvn(new.x,mu1,Sig1)+dmvn(new.x,mu2,Sig2))/Temps
        ivec<-log(runif(M)) < new.like-old.like
        old.x[ivec]<-new.x[ivec]
        old.like[ivec]<-new.like[ivec]

        #Exchange
        new.x<-old.x
        #Simple swap from target chain to tempered chain
        i<-1;j<-sample(2:M,size=1)
        new.x[i]<-old.x[j]
        new.x[j]<-old.x[i]
        new.like<-log(dmvn(new.x,mu1,Sig1)+dmvn(new.x,mu2,Sig2))/Temps
        if(log(runif(1)) < new.like[i]+new.like[j]-old.like[i]-old.like[j]){
                old.like[i]<-new.like[i]
                old.like[j]<-new.like[j]
                old.x[i]<-new.x[i]
                old.x[j]<-new.x[j]
        }
        if(iter > nburn & iter %% nthin == 0){
                ico<-ico+1
                x.samp[ico,,]<-old.x
                like.val[ico,]<-old.like
        }
}
par(pty='s',mar=c(4,4,2,2))
image.plot(xv,yv,fmat,col=colfunc(100),zlim=range(-200,-0),
           xlab=expression(x[1]),ylab=expression(x[2]),cex.axis=0.8)
contour(xv,yv,fmat,add=T,levels=seq(-150,0,by=5)); points(x.samp[,1,],pch=19,cex=0.5)
```




```
par(mfrow=c(1,2),mar=c(4,4,0,2))
plot(x.samp[,1,1],ylim=range(0,100),xlab='Iter',ylab=expression(x[1]),type='l',cex.axis=0.8)
plot(x.samp[,1,2],ylim=range(0,100),xlab='Iter',ylab=expression(x[2]),type='l',cex.axis=0.8)
```



Annealed Importance Sampling: *Annealed Importance Sampling* (AIS) uses a SIS approach, but with auxiliary densities **of the same dimension** defined by

$$p_{0j}(x) = c_j g_{0j}(x) = c_j \{g_0(x)\}^{1-\xi_j} \{g_n(x)\}^{\xi_j}$$

where p_0 is a "diffuse" distribution, and

$$0 = \xi_0 < \xi_1 < \cdots < \xi_n = 1.$$

We have a "path" from $p_{00}(x) \equiv p_0(x)$ to the target distribution

$$p_{0n}(x) \equiv \pi_n(x) = c_n g_n(x).$$

For $j = 1, \ldots, n-1$, let P_j be a Markov kernel with invariant distribution p_{0j}. The AIS algorithm proceeds as follows:

1. Sample x_1 from p_{00}, set $w_0 = 1/g_0(x_1)$.
2. For $j = 1, 2, \ldots, n-1$,
   (i) Sample x_{j+1} from $P_j(x_j, \cdot)$;
   (ii) Set
   $$w_j = w_{j-1} \frac{p_{0j}(x_j)}{p_{0j}(x_{j+1})} = w_{j-1} \frac{g_{0j}(x_j)}{g_{0j}(x_{j+1})}.$$
3. Return to 1. and repeat to produce N samples and weights

   $$x_n^{(1)}, \ldots, x_n^{(N)} \qquad\qquad w_n^{(1)}, \ldots, w_n^{(N)}$$

   where
   $$w_n = \frac{1}{g_0(x_1)} \times \frac{g_{01}(x_1)}{g_{01}(x_2)} \times \frac{g_{02}(x_2)}{g_{02}(x_3)} \times \cdots \times \frac{g_{0\,n-1}(x_{n-1})}{g_{0\,n-1}(x_n)} \times g_n(x_n).$$

Consider the **reverse kernel** P_j^R defined using Bayes Theorem as

$$P_j^R(x_{j+1}, x_j) = \frac{P_j(x_j, x_{j+1}) p_{0j}(x_j)}{p_{0j}(x_{j+1})} = \frac{P_j(x_j, x_{j+1}) g_{0j}(x_j)}{g_{0j}(x_{j+1})},$$

and let

$$g^\star(x_{1:n}) = g_n(x_n) \prod_{j=1}^{n-1} P_j^R(x_{j+1}, x_j \qquad\qquad g_0^\star(x_{1:n}) = g_0(x_1) \prod_{j=1}^{n-1} P_j(x_j, x_{j+1})$$

where $g_0^\star(x)$ is proportional to the AIS proposal joint density $p_0^\star(x)$. Define

$$w^\star(x_{1:n}) = \frac{g^\star(x_{1:n})}{g_0^\star(x_{1:n})}$$

as the importance sampling weight for the augmented sample

$$x_{1:n} = (x_1, \ldots, x_n)$$

generated from $p_0^\star$. We then have that

$$w^\star(x_{1:n}) = \frac{g_n(x_n) \prod_{j=1}^{n-1} P_j^R(x_{j+1}, x_j)}{g_0(x_1) \prod_{j=1}^{n-1} P_j(x_j, x_{j+1})} = \frac{g_n(x_n) \prod_{j=1}^{n-1} \frac{P_j(x_j, x_{j+1}) g_{0j}(x_j)}{g_{0j}(x_{j+1})}}{g_0(x_1) \prod_{j=1}^{n-1} P_j(x_j, x_{j+1})}$$

That is, all the terms in $P_j(\cdot, \cdot)$ cancel, and

$$w^\star(x) = \frac{1}{g_0(x)} \times \frac{g_{01}(x_1)}{g_{01}(x_2)} \times \frac{g_{02}(x_2)}{g_{02}(x_3)} \times \cdots \times \frac{g_{0\,n-1}(x_{n-1})}{g_{0\,n-1}(x_n)} \times g_n(x_n)$$

This is the identical weight to the one computed recursively above. Note also that for each $j = 1, \ldots, n-1$,

$$\int P_j^R(x_{j+1}, x_j)\, dx_j = 1$$

so the marginal distribution of x_n from

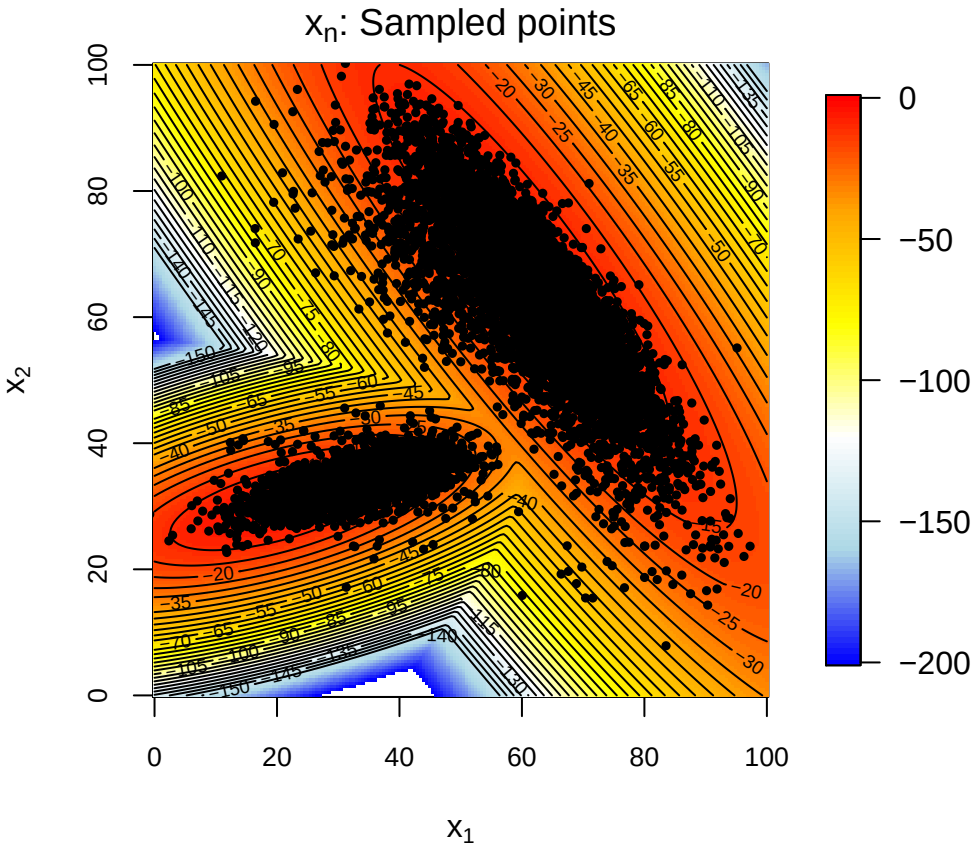
$$\pi^\star(x_{1:n}) = c^\star g^\star(x_{1:n})$$

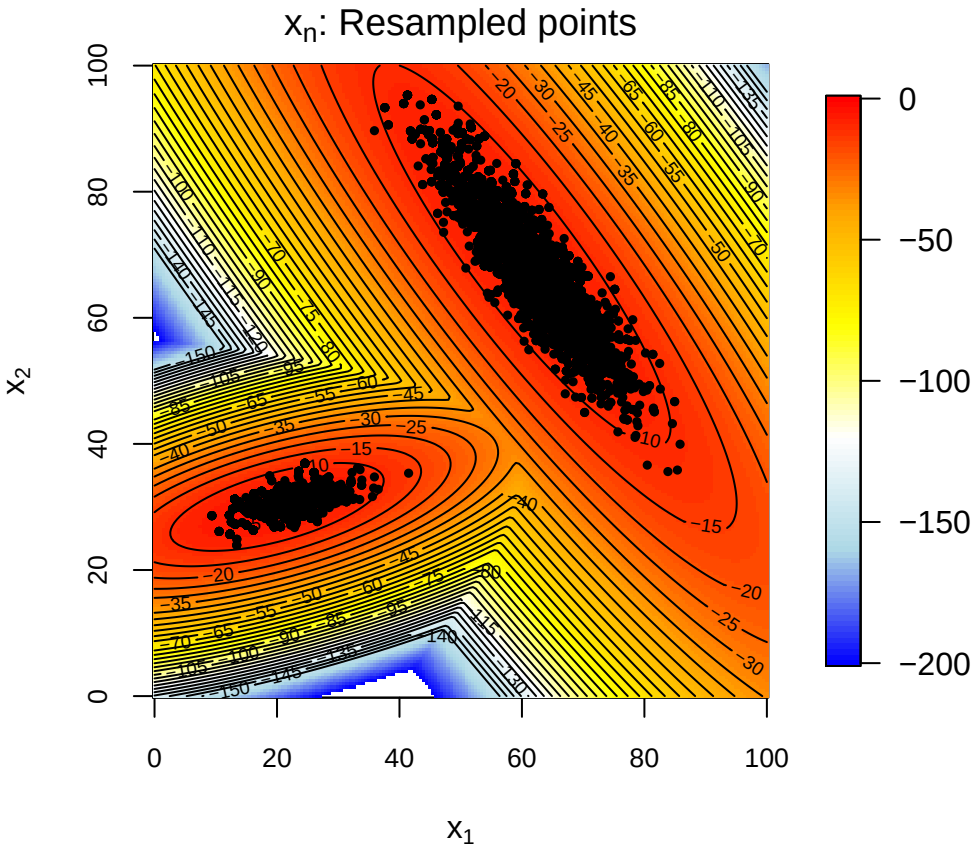
obtained by integrating $g^\star(x_{1:n})$ with respect to

$$x_1, x_2, \ldots, x_{n-1}$$

is precisely the true target $\pi_n(x_n)$.

```
p0j<-function(xv,m0,S0,m1,m2,S1,S2,Tmp){
    Tv<-(1-Tmp)*dmvn(xv,m0,S0,log=T)+Tmp*log(0.5*dmvn(xv,mu1,Sig1)+0.5*dmvn(xv,mu2,Sig2))
        return(Tv)
}

mu0<-c(50,50)
Sig0<-diag(c(200,200))

n<-20
set.seed(3764)
Temps<-c(1:n)/n
N<-10000
xmat<-array(0,c(n,N,2))
wmat<-matrix(0,nrow=n,ncol=N)
xmat[1,,]<-rmvn(N,mu0,Sig0)
old.like<-p0j(xmat[1,,],mu0,Sig0,mu1,mu2,Sig1,Sig2,0)
w0<--dmvn(xmat[1,,],mu0,Sig0,log=T)
for(j in 1:(n-1)){
    old.x<-xmat[j,,]
    new.x<-old.x+rmvn(N,c(0,0),diag(c(10,10)))
        old.like<-p0j(old.x,mu0,Sig0,mu1,mu2,Sig1,Sig2,Temps[j])
        new.like<-p0j(new.x,mu0,Sig0,mu1,mu2,Sig1,Sig2,Temps[j])
        ivec<-log(runif(N)) < (new.like-old.like)
        old.x[ivec]<-new.x[ivec]
        old.like[ivec]<-new.like[ivec]
    xmat[j+1,,]<-old.x
    wmat[j,]<-p0j(xmat[j,,],mu0,Sig0,mu1,mu2,Sig1,Sig2,Temps[j])-
              p0j(xmat[j+1,,],mu0,Sig0,mu1,mu2,Sig1,Sig2,Temps[j])
}
wmat[n,]<-p0j(xmat[n,,],mu0,Sig0,mu1,mu2,Sig1,Sig2,Temps[n])

wvec<-apply(wmat,2,sum)+w0
wvec<-exp(wvec-max(wvec))
wvec<-wvec/sum(wvec)

xsamp.sir<-xmat[n,sample(1:N,prob=wvec,size=N,rep=T),]
1/sum(wvec^2)

+ [1] 259.1652

par(pty='s',mar=c(4,4,2,2))
image.plot(xv,yv,fmat,col=colfunc(100),zlim=range(-200,-0),
           xlab=expression(x[1]),ylab=expression(x[2]),cex.axis=0.8)
contour(xv,yv,fmat,add=T,levels=seq(-150,0,by=5));
points(xmat[n,,],pch=19,cex=0.5)
title(expression(paste(x[n],': Sampled points')))
```


x_n: Sampled points
x_2
x_1
0
20
40
60
80
100
0
−50
−100
−150
−200


x_n: Resampled points
x_2
x_1
0
20
40
60
80
100
0
−50
−100
−150
−200