MCMC and some variations

Lorenz Wolf

December 9, 2021

Question 1

In this question we consider the following density on $\mathbb{R}^2$

$$f(x,y) = k \exp\left\{ -\frac{x^2}{100} - \left(y + \frac{3}{100}x^2 - 3 \right)^2 \right\}$$

where k is a normalising constant. First we will construct and implement an MCMC algorithm to sample from this distribution. We will then use this sampler to estimate the expected values of X and Y with joint distribution f(x,y). Finally, a sampling algorithm with adaptive proposal distribution as described by Haario et al. in [1] is implemented and then compared to the MCMC sampler.

a)

To sample from f(x,y) we propose a Metropolis Hastings algorithm with a additive random walk specified by a two dimensional normal distribution as outlined in Algorithm 1. We define

- Σ, a valid variance-covariance matrix of a 2-dimensional random variable
- nsteps, the number of iterations
- f(x,y), the target density up to a normalising constant
- $\mathcal{N}_2(\mu, \Sigma)$, the 2-d normal distribution with mean μ and variance covariance matrix Σ

Algorithm 1: MCMC algorithm with target f(x,y)

```
Inputs: $X_0 = (x_0, y_0)$, $\Sigma$, nsteps, f
for i=1...nsteps do
    Propose $Y \sim \mathcal{N}_2(X_{i-1}, \Sigma)$
    Simulate $U \sim \text{Uniform}(0,1)$
    if $U \leq \alpha = min\left(1, \frac{f(Y)}{f(X_{i-1})}\right)$ then
        $X_i = Y$
    else
        $X_i = X_{i-1}$
    end
end
```

In Figure 1 the unnormalised density is shown. It can be observed that X and Y have very different scales, i.e. the variance in X is much larger than in Y. Thus, running the algorithm for a simple standard proposal with Σ such as

$$\Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

will lead to poor mixing of the chain.
Since the mode of the distribution can be analytically computed to be $p^* = (x,y) = (0,3)$, we will use this as starting values for all future runs of the algorithm. To improve the mixing we need to take the different scales and dependency of X and Y into account by choosing Σ appropriately. In order to choose these values we iteratively use Algorithm 1 (starting with $\Sigma = I_1$) to sample from the target distribution, then compute sample covariance, and standard deviations to obtain a better estimate for Σ. This new estimate is used to specify the proposal in the next iteration. Afterwards we try several values for Σ around this estimate and obtain that the effective sample size is maximised for

$$\hat{\Sigma} = \begin{pmatrix} 7^2 & 0.22 \\ 0.22 & 2.1^2 \end{pmatrix}.$$

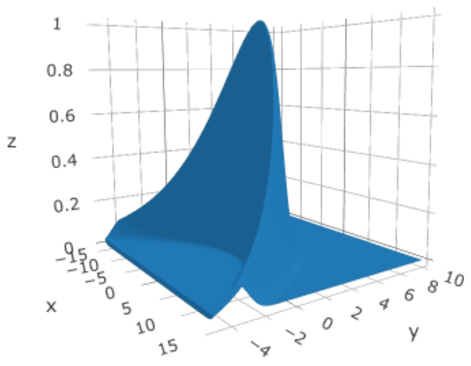

Figure 1: Joint density of X and Y.

To assess the convergence we run the algorithm for a sample size of 5×10^4 and $\Sigma = \hat{\Sigma}$ with 6 different starting values. The traceplots presented in Figures 2 and 3 show good convergence with occasional values of Y large and negative. Furthermore, we compute the Gelman Rubin diagnostic and obtain a point estimate and upper confident interval of (1,1.01) and (1,1.01) for X and Y respectively. Additionally, the gelman plots shown in Figures 4 and 5 do not indicate problems with convergence.

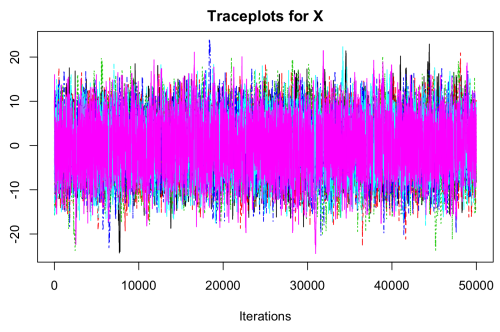

Figure 2: Traceplots of X for chain length 5×10^4 with 6 different starting values

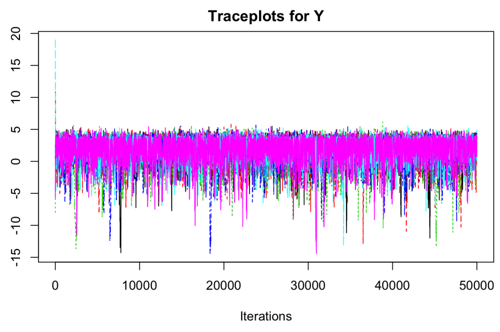

Figure 3: Traceplots of Y for chain length 5×10^4 with 6 different starting values

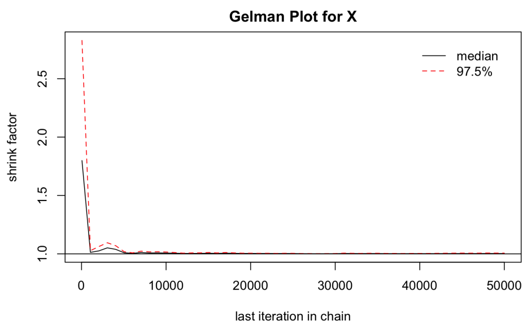

Figure 4: Gelman plot for X for chain lengths 5×10^4 with 6 different starting values.

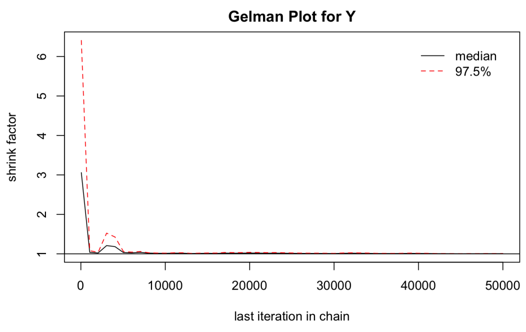

Figure 5: Gelman plot for Y for chain lengths 5×10^4 with 6 different starting values.

With $\Sigma = \hat{\Sigma}$ and starting value p^* the algorithm is run to obtain a sample of size 10^5. The first 10^4 values of this sample are omitted as burn-in to eliminate the influence of the starting value. In Figures 6 and 7 the acf's are presented. They show a reasonable decrease indicating that mixing is ocurring. We note that there is still significant dependence in the chain. However, having tried several different inputs for Σ no significant improvement has been found. Plotting the joint density in Figure 8 reveals the 'banana'-like shape required according to the plot of the real density in Figure 1. However, in the extreme regions the symmetry of f in x is not observed.

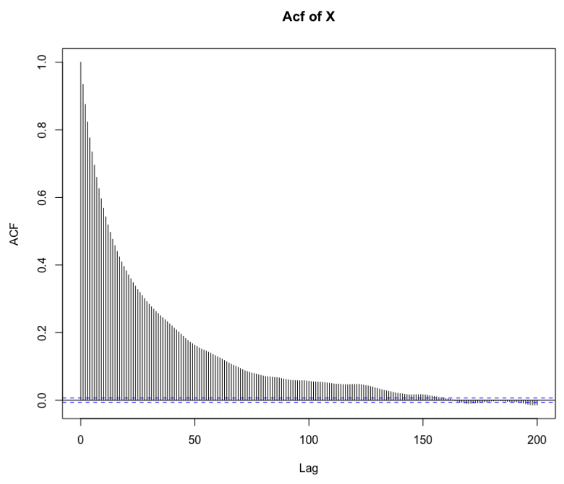

Figure 6: Acf of Y for sample size of 9×10^4 after burn-in with starting value (0,3), $\Sigma = \hat{\Sigma}$ and burn-in of 10^4.

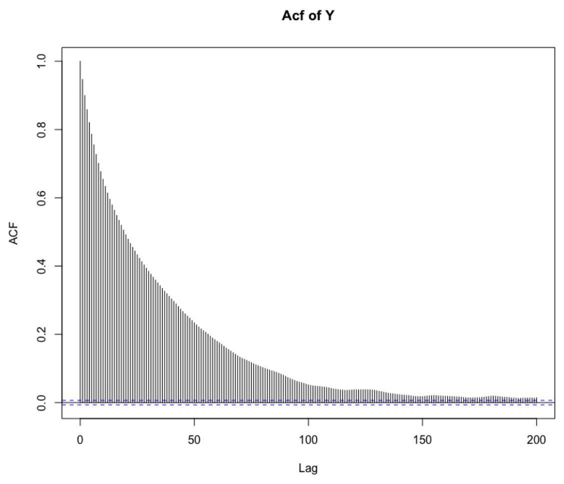

Figure 7: Acf of Y for sample size of 9×10^4 after burn-in with starting value (0,3), $\Sigma = \hat{\Sigma}$ and burn-in of 10^4.

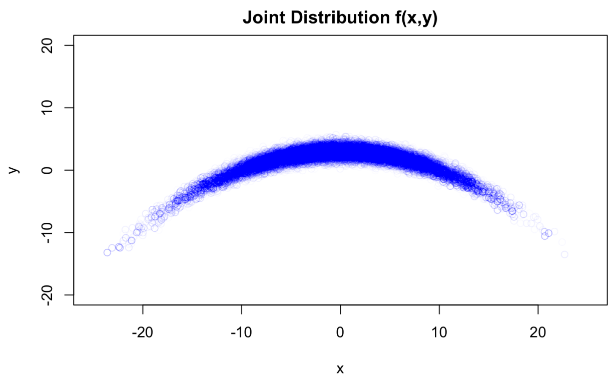

Figure 8: Joint density of X and Y for sample size of 9×10^4 after burn-in with starting value (0,3), $\Sigma = \hat{\Sigma}$ and burn-in of 10^4.

b)

Using the sampler discussed in part a) with starting value $X_0 = (0, 3)$ and $\Sigma = \hat{\Sigma}$ as specified in part a) we obtain a sample of size 9×10^4, after having discarded the first 10^4 points as burn-in. In the following computations of the means we use this sample. Due to the relatively high dependency in the resulting chain we suggest to use thinning. However, since the acf decreases slowly an appropriate step size for thinning would need to be large. As this would leave us with only few observations (limitations due to time and computational power to sample a longer chain) we omit thinning at this point and note that for better estimates a longer chain should obtained and thinned appropriately. The estimates obtained are

$$\hat{E}(X) = \frac{1}{9 \times 10^4} \sum_{t=1}^{9\times 10^4} X_i \approx -0.2817$$
$$\hat{E}(Y) = \frac{1}{9 \times 10^4} \sum_{t=1}^{9\times 10^4} Y_i \approx 1.5461$$

Using numerical integration the true values are computed to be $\mathbb{E}(X) = 0$ and $\mathbb{E}(Y) \approx 1.614$. Thus, the estimates (especially for X) are not very accurate with errors of 0.28 and 0.068 respectively.

c)

In this section we implement a random walk Metropolis algorithm with an adaptive proposal distribution as suggested in [1]. The key difference to the sampler in part a) is that the proposed adaptive algorithm adjusts the proposal distribution based on the samples so far obtained.

In particular assume the points $X_1, \dots, X_k$ have been already sampled. Then the proposal Y is sampled from the distribution $q_k(.|X_1, \dots, X_k)$ which depends on the last H observations $X_{k-H+1}, \dots, X_k$, where H is one of the input parameters to be specified. As proposal $q_k(.|X_1, \dots, X_k)$ a 2 dimensional Gaussian distribution with mean (X_k) and a covariance dependent on the sampled points is implemented. With at least H points sampled at time t define $K \in \mathbb{R}^{H \times d}$ to be the matrix with row i equal to X_{t-H+i} and define $\tilde{K} = K - \mathbb{E}(K)$ the corresponding centered matrix. Here $\mathbb{E}(K)$ is estimated by the mean of the H values. Then the proposal is taken to be

$$q_t(.|X_1, \dots, X_t) \sim X_t + \frac{c_d}{\sqrt{H-1}} \tilde{K}^T \mathcal{N}(0, I_H), \tag{1}$$

where I_H is the H dimensional identity matrix and $c_d = \frac{2.4}{\sqrt{d}}$.

To initialise the algorithm the first H points are sampled with the MCMC sampler from part a). Afterwards the proposal is updated as specified above and then further updates are performed according to the update frequency U. For an outline of the algorithm refer to Algorithm 2.

Algorithm 2: Adaptive Proposal Metropolis algorithm with target f(x,y)

```
Inputs: $X_0 = (x_0, y_0)$ (starting value), $\Sigma$ (inital covariance), H (History length), U (Update frequency),
        nsteps (Number of iterations), f (Target up to proportionality)
for $i$=1...$H$ do
    Propose $Y \sim \mathcal{N}_2(X_{i-1}, \Sigma)$
    Simulate $V \sim \text{Uniform}(0,1)$
    if $V \leq \alpha = min\left(1, \frac{f(Y)}{f(X_{i-1})}\right)$ then
        $X_i = Y$
    else
        $X_i = X_{i-1}$
    end
end
K = $(X_1, \dots, X_H)^T$
$\tilde{K} = K - E(K)$
for $i$=H+1...nsteps+H do
    Propose $Y \sim X_t + \frac{c_2}{\sqrt{H-1}} \tilde{K}^T \mathcal{N}(0, I_H)$
    Simulate $V \sim \text{Uniform}(0,1)$
    if $V \leq \alpha = min\left(1, \frac{f(Y)}{f(X_{i-1})}\right)$ then
        $X_i = Y$
    else
        $X_i = X_{i-1}$
    end
    if $i$ = 0 mod $U$ then
        K = $(X_{i-H+1}, \dots, X_i)^T$
        $\tilde{K} = K - E(K)$
    end
end
return $(X_{H+1}, \dots, X_{H+nsteps})$
```

To assess the convergence, the algorithm is run for 6 different starting values with U=H=200 (chosen according to [1]). The traceplots of the chains of length 5×10^4 for X and Y are shown in Figures 9 and 10 respectively and do not reveal any issues with convergence. Furthermore, the Gelman Rubin diagnostic and an upper confident interval are computed to be (1,1) for X and (1,1) for Y. Additionally, the Gelman plots are checked to look fine.

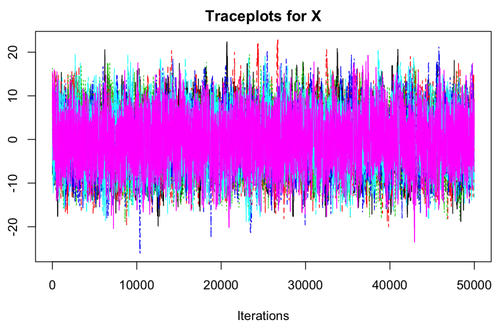

Figure 9: Traceplots of X for chain length 3×10^4 with 6 different starting values

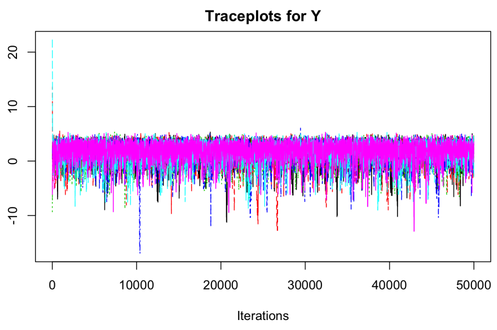

Figure 10: Traceplots of Y for chain length 3×10^4 with 6 different starting values

With U=H=200, starting value (0,3) and Σ as above we obtain a sample of size 9×10^4 after a burn-in of 10^4. The Acf's of X and Y for this sample are presented in Figures 11 and 12 respectively and indicate no problems. The joint density has the 'banana'-like shape as required.

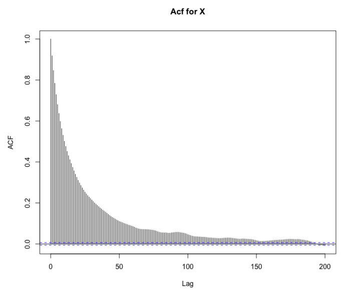

Figure 11: Acf of X for adaptive proposal sampler.

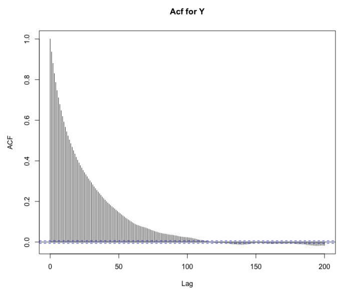

Figure 12: Acf of Y for adaptive proposal sampler.

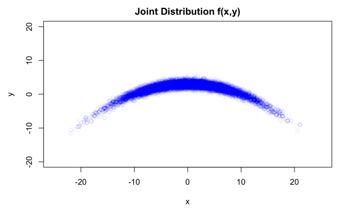

Figure 13: Joint density of X and Y

Advantages compared to sampler a)

Haario et al describe the specification of a proposal as the potential bottleneck of the Metropolis algorithm when we have small knowledge of the target distribution. The proposal distribution influences the mixing of the chain and the convergence. Furthermore, it should be easy to sample from for computational efficiency. The adaptive proposal addresses the problem of choosing a proposal by automatically adjusting the proposal distribution based on the samples already obtained. The adaptive proposal proves to be especially strong when the target distribution is non-linear or curved and there is correlation between the components. In the case of correlated components Haario et al. have found the adaptive proposal to be more effective compared to a Metropolis algorithm with not well tuned proposal and not much less effective compared to a Metropolis algorithm with optimally selected proposal. Compared to the sampler in part a) with an adaptive proposal we do not have to be concerned with the choice of Σ which can prove to be difficult. The implementation of the sampler in part c) shows increased computational speed and smaller upper confident intervals for the Gelman Rubin diagnostic. Furthermore, acf's for the adaptive proposal sampler shown in Figures 11 and 12 decrease quicker than those for the sampler in part a), which indicates better mixing and smaller dependence in the chain. Additionally we obtain increased effective sample sizes compared to the sampler in part a) as shown in Table 1. The resulting estimates of the means are also more accurate. Comparing the joint densities shown in Figures 8 and 13 it can be observed that the properties of the true joint density (e.g. symmetry) for extreme x-values are more accurate for the sample obtained with the adaptive proposal sampler. All these are desirable properties of a sampler which are obtained, without choosing a specific proposal. A disadvantage that is not observed in our case is that the adaptive proposal sampler is an approximate method which can lead to a bias

in the simulation [1].
For a more detailed comparison of the samplers the bias and standard errors as well as root mean squared errors of several estimates could be computed by repeated sampling with the different samplers.

		ESS	$\hat{\mathbb{E}}(.)$	Time
Sampler a)	X	1883.4	-0.2817	1.34s
	Y	1409.6	1.5461	
Sampler b)	X	2374.1	0.021	1.22s
	Y	1989.8	1.622	

Table 1: Properties based on samples of size 10^5 and starting values $X_0 = c(0,3)$ and $\Sigma = \hat{\Sigma}$. For the adaptive proposal we take U=H=200. A burn-in of 10^4 points is chosen. Time reported is the average time of 10 iterations taken to sample a chain of length 5×10^4

Question 2

In Question 2 we consider the density

$$g(\theta) = k\left(\frac{4}{10}\exp\left\{-(8-\theta)^2\right\} + \frac{6}{10}\exp\left\{-(30-\theta)^2\right\}\right) \propto \exp\left\{-H(\theta)\right\},$$

where k is the normalising constant. In Figure 16 we can observe the two separated modes of g(θ) (Tempered distribution with T=1).

a)

First consider an additive random walk Metropolis Hastings algorithm with normally distributed noise (mean 0 and variance σ^2) to sample from g(.). To demonstrate that such an algorithm will not explore the entire state space we consider two different scenarios.

First, consider such an algorithm with small *sigma* (e.g. around 1). With such a σ, the acceptance rate will be good enough for the states to change frequently. However, the variance of the proposal is to small for the chain to jump between the different modes of g(.) (except in some very rare cases). Thus, depending on the starting value a chain converges to one of the modes and does not explore the entire state space. This scenario can be observed in Figure 14.
Secondly consider such an algorithm with large σ. Due to the large variance the proposals are more likely to be far away from the current state. Thus the chain might jump between the different modes. However, such proposals are less likely to be accepted, so the chain jumps fewer times. This causes very high dependence and slow to no convergence of the chain. This scenario can be observed in Figure 15.
We conclude that the entire state space can not be explored properly with a simple Metropolis Hastings algorithm.

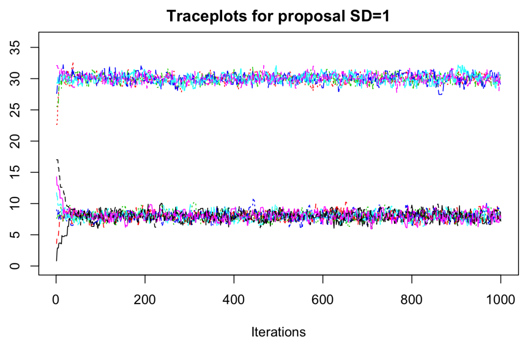

Figure 14: Traceplots for chains of length 10^3 for several starting values with proposal standard deviation equal to 1.

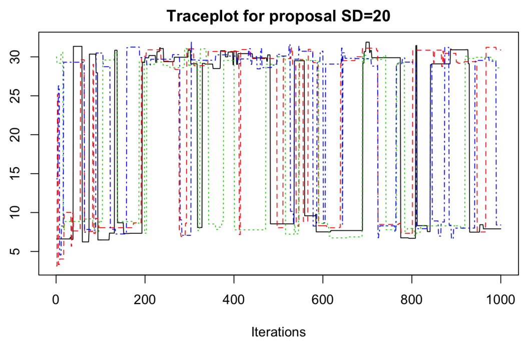

Figure 15: Traceplots for chains of length 10^3 for several starting values with proposal standard deviation equal to 20.

b)

To sample from this density a parallel tempering algorithm is implemented. We define the tempered distributions with temperatures T_i, i=1,...,M, to be

$$g_{T_i}(\theta) = k\left(\frac{4}{10}\exp\left\{-(8-\theta)^2\right\} + \frac{6}{10}\exp\left\{-(30-\theta)^2\right\}\right)^{1/T_i}\frac{1}{Z_{T_i}} \propto \exp\left\{\frac{-H(\theta)}{T_i}\right\},$$

where $Z_{T_i} = \int g_{T_i}(\theta)d\theta$ and $T_1 < T_2 < \cdots < T_M$.

One iteration of the implemented parallel tempering algorithm consists of a normal additive random walk proposal Metropolis Hastings step to update each of the M chains, which upon setting the variance of the proposal distribution for each chain fixes the transition kernels $P_1, \ldots, P_M$ (for details refer to Algorithm 3). After each chain is updated, swaps between the chains are proposed. For each chain l=1,...,M we define the between-chain exchange probabilities according to which swaps are proposed to be

$$\begin{aligned} q_B(l,m) &= \text{P[Swap chain } l \text{ with chain } m \mid \text{ chain } l \text{ selected]} \\ &= \begin{cases} \frac{1}{2} & 2 \le I < M, m = l-1, l+1 \\ 1 & l = 1 \text{ and } m = 2, \text{ or } l = M \text{ and } m = M-1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Then the between-chain exchange that proposes to swap x_l,the current state of chain l, with x_m, that of chain m, is accepted with probability

$$\alpha = \min\left\{1, \frac{\pi_l(x_m)\pi_m(x_l)q_B(m,l)}{\pi_l(x_l)\pi_m(x_m)q_B(l,m)}\right\}$$

For the tempered distributions

$$\frac{\pi_l(x_m)\pi_m(x_l)}{\pi_l(x_l)\pi_m(x_m)} = \exp\left\{(H(x_l) - H(x_m))\left(\frac{1}{T_l} - \frac{1}{T_m}\right)\right\}$$

The samples collected for x_1 across iterations represent a sample from the marginal $\pi_1 \equiv \pi$, the target distribution. In this particular implementation outlined in Algorithm 3 we propose swaps in a random order each iteration. For cleaner code we define a function called $rq_B(.)$ proposing the swapping partner given chain l is selected according to $q_B(l,.)$. Furthermore, the Metropolis steps are vectorised but will be written as a for loop in the pseudo code.

Algorithm 3: Parallel tempering algorithm for g(θ)

```
Inputs: $\mathbf{X_0} = (\mathbf{X_0^{(1)}}, \ldots, \mathbf{X_0^{(M)}})$ (vector of starting values), Ts (vector of temperatures),
        $\sigma = (\sigma^{(\mathbf{1})}, \ldots, \sigma^{(\mathbf{M})})$ (proposal standard deviations for each temperature), nsteps (Number of
        iterations), g (Target up to proportionality)
for i=1...nsteps do
    for l=1...M do
        Propose $Y \sim \mathcal{N}(X_{i-1}^{(l)}, \sigma^{(l)})$
        Simulate $V \sim$ Uniform(0,1)
        if $V \le \alpha = min\left(1, \frac{g(Y)}{g(X_{i-1}^{(l)})}\right)$ then
            $X_i^{(l)} = Y$
        else
            $X_i^{(l)} = X_{i-1}^{(l)}$
        end
    end
    for l in permutation({1,...,M}) do
        Propose swapping partner $m \sim rq_B(l,.)$
        Simulate $V \sim$ Uniform(0,1)
        if $V \le \alpha = min\left(1, \exp\left\{(H(x_l) - H(x_m))\left(\frac{1}{T_l} - \frac{1}{T_m}\right)\right\}\frac{q_B(m,l)}{q_B(l,m)}\right)$ then
            Swap $X_i^{(l)}$ with $X_i^{(m)}$
        end
    end
end
```

For the following discussion we use temperatures $Ts = (1, 2, 5, 10, 20, 50, 100)$ with transition kernels specified by the standard deviations $\sigma = (0.3, 0.6, 1, 1.6, 4, 9, 14)$. The variances are chosen increasingly as with increasing temperature the tempered distributions become flatter so that big jumps are still accepted. The swaps will then accelerate the mixing. In Figure 16 the shapes of some tempered distributions can be observed. Figure 17 shows the traceplots for 6 distinct starting values and 10^4 iterations indicating good convergence. Furthermore the Gelman plot is presented in Figure 18 and the Gelman Rubin diagnostic is computed to be 1 with upper confident interval 1.

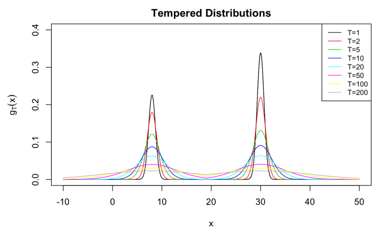

Figure 16: Tempered distributions

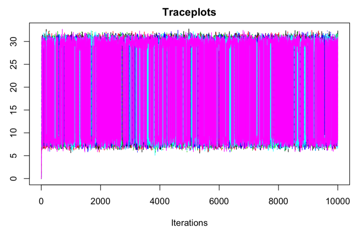

Figure 17: Traceplots

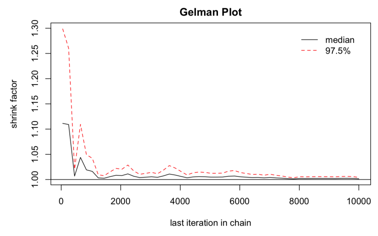

Figure 18: Gelman plot

Taking starting the starting value $X_0 = 0$ for each chain and with the transition kernels as specified above we obtain a sample of length 9×10^4 after the first 10^4 points are discarded as burn-in. Figure 20 shows the acf of this sample. The acf decreases relatively quickly which indicates good mixing and hence good exploration of the state space. The ESS is computed to be 4316. The Histogram presented in Figure 20 matches the shape of that of the true density well. For a better comparison the estimated density based on the sample is plotted together with the true density in Figure 21. The two densities are a very close fit with small discrepancies in the modes. Thus, we can conclude that the sampler is exploring the full state space and correctly sampling from the target distribution.

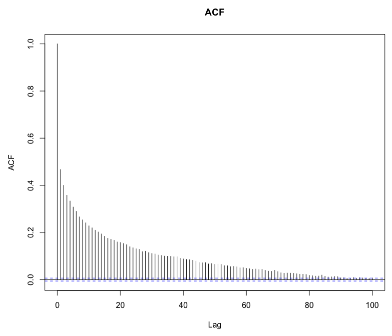

Figure 19: Acf of sample

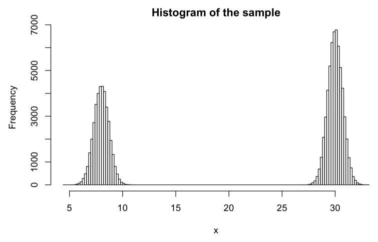

Figure 20: Histogram of sample

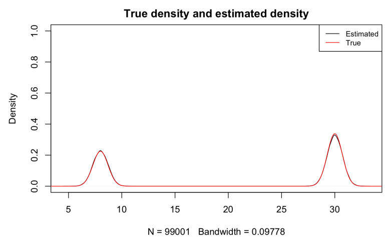

Figure 21: Estimated vs true density

c)

In this part we estimate $P(\theta > 20)$ where $\theta \sim g(.)$. The true value is numerically computed to be 0.6 with absolute error smaller 1.6×10^{-5}. Based on the sample obtained in the way as described above we compute the estimate

$$\hat{P}_{20} = \frac{1}{9 \times 10^4} \sum_{i=1}^{9\times 10^4} I(X_i > 20) = 0.6005$$

which is accurate with an error of 5×10^{-4}.

References

[1] Haario & Saksman & Tamminen. "Adaptive proposal for random walk Metropolis algorithm." In: *Computational Statistics* 14 (1999), pp. 375–395.

Appendix

Listing 1: Code for question 1

```r
# plot density
f <- function(x,y) exp(-x^2/100 - (y+3/100*x^2-3)^2)

x_vec <- seq(-15,15,0.1)
y_vec <- seq(-5,10,0.1)
grid <- expand.grid(x=x_vec,y=y_vec)
f_points <- f(grid$x, grid$y)

dat <- data.frame(x = c(grid$x), y=c(grid$y), f=c(f_points))

fig <- plot_ly(x = dat$x, y=dat$y, z=dat$f, type = 'mesh3d')
fig
# simple metropolis hastings
MetrHastw <- function(x0, sigmapropx, sigmapropy, nsteps, f){
  X <- matrix(NA,nrow=2, ncol=nsteps+1)
  X[,1] <- x0

  for (i in 2:(nsteps+1)){
    x <- rnorm(1,mean=X[1,(i-1)], sd=sigmapropx)
    y <- rnorm(1,mean=X[2,(i-1)], sd=sigmapropy)
    if (runif(1)<=min(exp(log(f(x,y))-log(f(X[1,i-1], X[2,i-1]))), 1)) X[,i] <- c(x↩
        ,y)
    else X[,i] <- X[,i-1]
  }
  X[,-1]
}
# algorithm taking into account scale and dependency
# by specifying the covariance matrix of the proposal
MetrHastw_cov <- function(x0, sigmapropx, sigmapropy, cova, nsteps, f){
  X <- matrix(NA,nrow=2, ncol=nsteps+1)
  X[,1] <- x0

  for (i in 2:(nsteps+1)){
    prop <- rmvnorm(1,mean=X[,i-1], sigma=matrix(c(sigmapropx^2,cova,cova,↩
        sigmapropy^2), ncol=2))
    x <- prop[1]
    y <- prop[2]
    if (runif(1)<=min(exp(log(f(x,y))-log(f(X[1,i-1], X[2,i-1]))), 1)) X[,i] <- c(x↩
        ,y)
    else X[,i] <- X[,i-1]
  }
  X[,-1]
}


# tuning parameters
X_mh_adj <- MetrHastw_cov(c(0,3), 6.884244,2.122618, 0.2264226,10^5,f)
cov <- cov(X_mh_adj[1,], X_mh_adj[2,])
sd_x <- sd(X_mh[1,])
sd_y <- sd(X_mh[2,])
cov <- 0
sd_x <- 1
sd_y <- 1

for (i in 1:5){

  print(c(sd_x, sd_y,cov))
  Dummy <- MetrHastw_cov(c(0,3), sd_x,sd_y, cov,10^5,f)
  # update parameters
  cov <- cov(Dummy[1,10^4:10^5], Dummy[2,10^4:10^5])
  sd_x <- sd(Dummy[1,10^4:10^5])
  sd_y <- sd(Dummy[2,10^4:10^5])

}
# Then trying several values around this estimate
# multiple chain convergence diagnostics
set.seed(0)
N <- 5*10^4 # chain length
X0s <- replicate(6, rnorm(2,c(0,3), 8))
samplesX <- matrix(rep(0,dim(X0s)[2]*N), nrow=dim(X0s)[2])
samplesY <- matrix(rep(0,dim(X0s)[2]*N), nrow=dim(X0s)[2])
for (i in 1:dim(X0s)[2]){
  sample <- MetrHastw_cov(X0s[,i], 7 ,2.1, 0.22,N,f)
  samplesX[i,] <- sample[1,]
  samplesY[i,] <- sample[2,]
}


chainsX <- lapply(1:dim(X0s)[2], function(i) mcmc(samplesX[i,]))
chainsY <- lapply(1:dim(X0s)[2], function(i) mcmc(samplesY[i,]))
X0s
traceplot(chainsX, main='Traceplots for X')
traceplot(chainsY, main='Traceplots for Y')
gelman.diag(mcmc.list(chainsX))
gelman.diag(mcmc.list(chainsY))
gelman.plot(mcmc.list(chainsX), main='Gelman Plot for X')
gelman.plot(mcmc.list(chainsY), main='Gelman Plot for Y')

# sample
set.seed(0)
X_mh_final <- MetrHastw_cov(c(0,3), 7 ,2.1, 0.22,10^5,f)
plot(X_mh_final[1,], type='l')
plot(X_mh_final[2,], type='l')

plot(X_mh_final[1,(500:10^5)], X_mh_final[2,(500:10^5)], col=rgb(red=0, green=0, blue=1, alpha=0.01), xlab='x', ylab='y', main='Joint Distribution f(x,y)', xlim=c(-25,25), ylim=c(-20,20))
acf(X_mh_final[1,10^4:10^5], lag.max=200, main='Acf of X')
acf(X_mh_final[2,10^4:10^5], lag.max=200, main='Acf of Y')

#mean estimates
mean(X_mh_final[1,(10^4+1):10^5])
mean(X_mh_final[2,(10^4+1):10^5])

#numerical integration for finding means
IntY <- function(x) sapply(x, function(b) integrate(function(y) f(b,y),-Inf,Inf)$value)
IntX <- function(y) sapply(y, function(b) integrate(function(x) f(x,b),-Inf,Inf)$value)
k <- 1/integrate(function(x) IntY(x), -Inf, Inf)$value

k*integrate(function(x) x*IntY(x), -Inf, Inf)$value
k*integrate(function(y) y*IntX(y), -Inf, Inf)$value

# adaptive proposal algorithm
c_2 <- 2.4/sqrt(2)
AP <- function(X0=c(0,3), H, U, nsteps, f){

  X <- matrix(NA,nrow=2, ncol=nsteps+H)
  X[,1] <- X0
  X[,2:(H)] <- MetrHastw_cov(X0, 7, 2, 0.22, H-1, f)

  Kt <- X[,1:H]
  Kt_centered <- Kt - apply(Kt, 1, mean)

  for (i in (H+1):(nsteps+H)){

    Y <- X[,i-1] +  c_2/sqrt(H-1)* Kt_centered  %*% rnorm(H)

    if (runif(1)<=min(f(Y[1], Y[2])/f(X[1,i-1], X[2,i-1]), 1)) X[,i] <- Y
    else X[,i] <- X[,i-1]

    if ((i%%U)==0){
      Kt <- X[,(i-H+1):(i)]
      Kt_centered <- Kt - apply(Kt, 1, mean)
    }
  }
  X[,(1+H):(H+nsteps)]
}

# ap multiplc chain conv diagnostics
set.seed(0)
N <- 5*10^4 # chain length
X0s <- replicate(6, rnorm(2,c(0,3), 8))
samplesX_ap <- matrix(rep(0,dim(X0s)[2]*N), nrow=dim(X0s)[2])
samplesY_ap <- matrix(rep(0,dim(X0s)[2]*N), nrow=dim(X0s)[2])
for (i in 1:dim(X0s)[2]){
  sample_ap <- AP(X0s[,i],H=200, U=200, nsteps=N, f)
  samplesX_ap[i,] <- sample_ap[1,]
  samplesY_ap[i,] <- sample_ap[2,]
}


chainsX_ap <- lapply(1:dim(X0s)[2], function(i) mcmc(samplesX_ap[i,]))
chainsY_ap <- lapply(1:dim(X0s)[2], function(i) mcmc(samplesY_ap[i,]))
X0s
traceplot(chainsX_ap, main='Traceplots for X')
traceplot(chainsY_ap, main='Traceplots for Y')
gelman.diag(mcmc.list(chainsX_ap))
gelman.diag(mcmc.list(chainsY_ap))
gelman.plot(mcmc.list(chainsX_ap), main='Gelman Plot for X')
gelman.plot(mcmc.list(chainsY_ap), main='Gelman Plot for Y')

# sample
set.seed(0)
X_AP2 <- AP(c(0,3),H=200, U=200, nsteps=10^5, f)
plot(X_AP2[1,(10^4:10^5)], X_AP2[2,(10^4:10^5)], col=rgb(red=0, green=0, blue=1, ↩
    alpha=0.01), xlab='x', ylab='y', main='Joint Distribution f(x,y)', xlim=c↩
    (-25,25), ylim=c(-20,20))
acf(X_AP2[1,(10^4:10^5)], lag.max=200, main='Acf for X')
acf(X_AP2[2,(10^4:10^5)], lag.max=200, main='Acf for Y')
plot(X_AP2[1,(10^4:10^5)], type='l')
plot(X_AP2[2,(10^4:10^5)], type='l')



#effective sample size and estimates
mean(X_AP2[1,10^4:10^5])
mean(X_AP2[2,10^4:10^5])

ESS_ap2 <- c(effectiveSize(mcmc(X_AP2[1,10^4:10^5])), effectiveSize(mcmc(X_AP2↩
    [2,10^4:10^5])))
ESS_ap2
ESS_mh <- c(effectiveSize(mcmc(X_mh_final[1,10^4:10^5])), effectiveSize(mcmc(X_mh_↩
    final[2,10^4:10^5])))
ESS_mh
# time algorithms
system.time(replicate(10, AP(c(0,3),H=200, U=200, nsteps=5*10^4, f)))
system.time(replicate(10,MetrHastw_cov(c(0,3), 7 ,2.1, 0.22,5*10^4,f)))
```

Listing 2: Code for question 2

```
a <- 8
g <- function(theta) 4/10*exp(-(a-theta)^2) + 6/10 * exp(-(30-theta)^2)

x <- seq(0,38,0.01)
plot(x,g(x), type='l', main='g(.) up to proportionality')

MetrHastw1 <- function(X0, sigmaprop, nsteps, f){
  X <- numeric(nsteps+1)
  X[1] <- X0

  for (i in 2:(nsteps+1)){
    Y <- rnorm(1,mean=X[i-1], sd=sigmaprop)
    if (runif(1)<=min(f(Y)/f(X[i-1]), 1)) X[i] <- Y
    else X[i] <- X[i-1]
  }
  X[-1]
}
# not exploring entire state space
X0s <- seq(0,35,3)
chains <- lapply(1:length(X0s), function(i) mcmc(MetrHastw1(X0s[i], 1, 10^3, g)))
traceplot(chains, ylim=c(0,36), main='Traceplots for proposal SD=1')

X0s <- c(1,17,30,40)
chains <- lapply(1:length(X0s), function(i) mcmc(MetrHastw1(X0s[i], 20, 10^3, g)))
traceplot(chains, main='Traceplot for proposal SD=20')

# parallel tempering algorithm
g_temp <- function(theta,T) g(theta)^(1/T)
H <- function(x) -log(g(x))
# swapping probabilities
q_B <- function(l,m,M) {
  if(2<=l & l<M & (m==(l-1) | m==l+1)) return(0.5)
  if((l==1 & m==2) | (l==M & m==M-1)) return(1)
  else return(0)
}
rq_B <- function(l,M){
  if(l==1) return(l+1)
  if(l==M) return(l-1)
  if(2<=l & l<M) return(ifelse(runif(1)<1/2, l-1, l+1))
}

tempered <- function(Ts,x0s, sigmaprops, nsteps){
  M <- length(Ts)
  X <- matrix(rep(0,M*(nsteps+1)), nrow=M, ncol=nsteps+1)
  X[,1] <- x0s
  accepted_mh <- 0
```

```r
  accepted_sw <- 0
  # prob accept swap
  for (i in 2:nsteps+1){
    #proposal
    prop <- X[,i-1] + rnorm(M,mean=rep(0,M), sd=sigmaprops)
    U <- runif(M)
    ind <- c(U<= sapply(1:M, function(j) exp(log(g_temp(prop[j],Ts[j]))- log(g_temp(X[j,i-1],Ts[j])))))
    # mh update for each density
    X[ind,i] <- prop[ind]
    X[!ind,i] <- X[!ind, i-1]
    accepted_mh <- accepted_mh + sum(ind)
    # between moves
    for(l in sample(1:M)){
      m <- rq_B(l,M)
      if (runif(1)<=min(1, exp((H(X[l,i])-H(X[m,i]))*(1/Ts[l]-1/Ts[m])))) {
        X[c(l,m),i] <- X[c(m,l),i]
        accepted_sw <- accepted_sw + 1}
    }
  }
  print(accepted_mh/(nsteps*M))
  print(accepted_sw/(nsteps*M))
  return(X[,-1])
}

#plot tempered distributions
Ts <- c(1,2,5,10,20, 50, 100, 200)
x <- seq(-10,50,0.01)
k <- 1/integrate(g,lower=-Inf,upper=Inf)$value
plot(x,g(x)*k, type='l', main='Tempered Distributions', xlim=c(-10,50), ylim=c(0,.4), ylab=expression(g[T](x)))
for(i in 1:length(Ts)) lines(x, g_temp(x,Ts[i])/integrate(function(y) g_temp(y,Ts[i]), lower=-Inf, upper=Inf)$value, col=i)
legend('topright', legend=c("T=1", "T=2","T=5","T=10","T=20","T=50","T=100","T=200"),
       col=1:(length(Ts)), lty=1, cex=0.8)

# multiple chains convergence diagnostics
X0s <- replicate(6, rnorm(length(Tsq), 0 ,10))
chains <- lapply(1:6, function(i) mcmc(tempered(Tsq, X0s[,i], c(0.3, 0.6, 1, 1.6, 4, 9,14), 10^4)[1,]))
gelman.diag(mcmc.list(chains))
traceplot(chains, main='Traceplots')
gelman.plot(mcmc.list(chains), main='Gelman Plot')
# sample
set.seed(0)
Tsq <- c(1,2, 5, 10,20, 50, 100)
X_tempq <- tempered(Tsq, rep(0,length(Tsq)), c(0.3, 0.6, 1, 1.6, 4, 9,14), 10^5)
hist(X_tempq[1,10^3:10^5], breaks=150, main='Histogram of the sample', xlab='x')
plot(X_tempq[1,10^3:10^5], type='l', main='Traceplot', ylab=expression(X[t]))
acf(X_tempq[1,10^3:10^5], lag.max=100, main='ACF')

#plot true density against estimated density
x <- seq(0,35,0.1)
plot(density(X_tempq[1,10^3:10^5], adjust=0.1 ),ylim=c(0,1), main='True density and estimated density')
lines(x, g(x)*k, col='red')
legend('topright', legend=c("Estimated", "True"),
       col=c('black', 'red'), lty=1, cex=0.8)

# estimate and numerical integration
```

```r
mean(X_tempq[1,10^4:10^5]>20)
integrate(function(x) k*g(x), lower=20, upper=Inf)
```