Computational Statistics

Final coursework

Robin Mathelier, Imperial College London

7th-11th december 2020

Question 1

We consider the following density on $\mathbb{R}^2$:

$$f(x,y) = k \exp\left\{ -\frac{x^2}{100} - \left(y + \frac{3}{100}x^2 - 3 \right)^2 \right\} \tag{1}$$

where k is a normalising constant.

(a) We aim to construct an MCMC algorithm to sample from the density f.
First of all, we calculate the modes of the density f by computing $\frac{\partial f(x,y)}{\partial x}$ and $\frac{\partial f(x,y)}{\partial y}$. Setting the derivatives equal to 0 and noticing that $\forall (x,y) \in \mathbb{R}^2, f(x,y) > 0$, it is observed that the density f has only one mode given by :

$$(x^*, y^*) = (0,3) \tag{2}$$

From this, we can chose to construct a Metropolis-Hasting algorithm with a random walk proposal distribution. The new state $(\tilde{X}^t, \tilde{Y}^t)$ is proposed in function of the old state (X^{t-1}, Y^{t-1}) by :

$$\begin{pmatrix} \tilde{X}^t \\ \tilde{Y}^t \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} X^{t-1} \\ Y^{t-1} \end{pmatrix}, \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix} \right) \tag{3}$$

Let denote $\Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$, we give the pseudo-code of the Metropolis-Hasting algorithm for this random walk proposal distribution which we notice to be symmetric (Algorithm 1).

Algorithm 1 Metropolis-Hasting algorithm

```
1: Given initial values $(X^0, Y^0)$, the unnormalized target density $f$ as defined in equation 1, the covariance
   $2 \times 2$ matrix $\Sigma$, the number or iterations $N$.
2: for $t = 1, 2, ..., N$ do :
3:     $(\tilde{X}, \tilde{Y}) \sim \mathcal{N}((X^{t-1}, Y^{t-1}), \Sigma)$
4:     $U \sim U(0,1)$
5:     if: $U \leq \min\{\exp(\log(f(\tilde{X}, \tilde{Y})) - \log(f(X^{t-1}, Y^{t-1})), 1\}$ then:
6:         $(X^t, Y^t) = (\tilde{X}, \tilde{Y})$
7:     else:
8:         $(X^t, Y^t) = (X^{t-1}, Y^{t-1})$
```

We aim to tune the values of σ_x, σ_y and ρ in order to obtain a good mixing for the markov chains sampled from the Metropolis-Hasting algorithm. To do so, we run the latter for a grid of values of σ_x, σ_y and ρ and we compute the effective sample size of the generated samples. At this stage, the Metropolis-Hasting algorithm is used for $n = 5000$ iterations with a starting value $(X^0, Y^0) = (0,3)$. We first use different scales for the parameters (from 10^{-3} to 10^1), and we pick up the values of the parameters that maximize the effective sample sizes. Then, we create another grid centered on the obtained values. We repeat the process and we choose the following values that give an effective sample size equal to 1264 for $(X^t)_{t=1,..,N}$ and 584 for $(Y^t)_{t=1,..,N}$ with $N = 10^5$ iterations :

$$\sigma_x = 8.33 \qquad \sigma_y = 1.33 \qquad \rho = 0.05 \tag{4}$$

It is noticed that the effective sample sizes are not very good, the target distribution will not be sampled very well. This can be improved running the grid search for more iterations and more precise tuning of the parameters but this requires time and intense computational ressources. We run the Metropolis-Hastings algorithm (Algorithm 1) with the appropriate values (Equation 4) for the covariance matrix. We obtain realisations of a Markov chain $\left(X^t, Y^t\right)_{t=1,..,N}$ with stationary distribution f.

We plot the traceplots of the samples $\left(X^t, Y^t\right)_{t=1,..,N}$ obtained for different initial values (Table 1). The markov chains seem to converge for each initial value and to explore well the space. Moreover, we compute a gelman coefficient equal to 1 for $\left(X^t\right)_{t=1,..,N}$ (upper 95% confidence interval equal to 1.01) and equal to 1.01 (upper 95% confidence interval equal to 1.01) for $\left(Y^t\right)_{t=1,..,N}$ which indicate a good behaviour and a convergence of our sampled chains. The traceplots suggest to do a burn-in of the first 2000 samples (See dotted line in figure 1).

i	1	2	3	4	5
X_i^0	-20.00	-10.00	0.00	10.00	20.00
Y_i^0	-20.00	-10.00	0.00	10.00	20.00

Table 1: Initial values chosen for the Markov chains $\left(X^t, Y^t\right)_{t=1,..,N}$

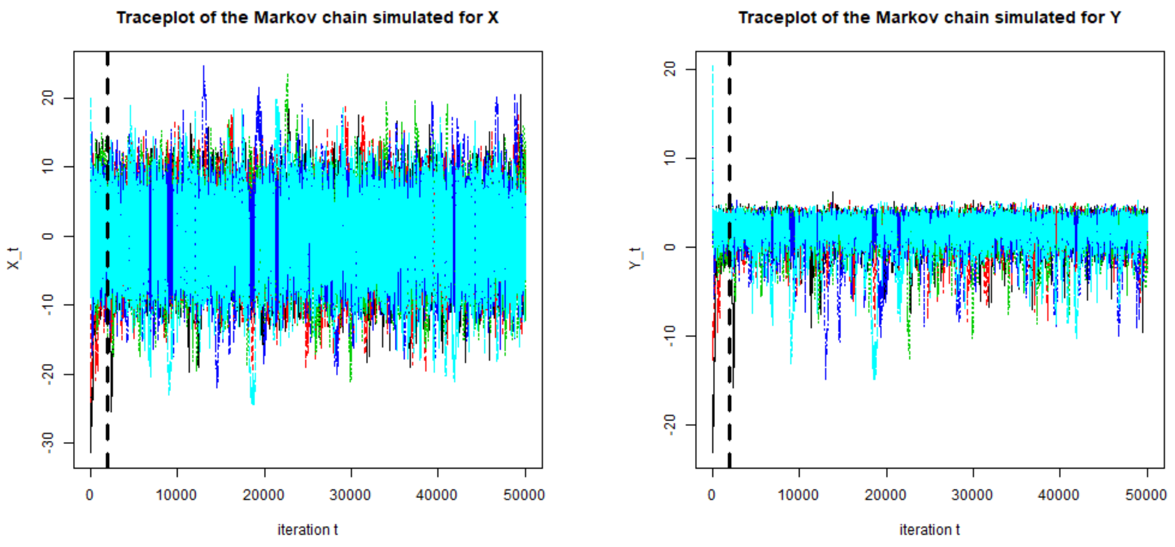

Figure 1: Traceplots of the samples $\left(X^t, Y^t\right)_{t=1,..,N}$ for different initial values with MH algorithm

We also give an autocorrelation plot for X and Y (Figure 2) for an initial value $(X^0, Y^0) = (0, 3)$. The decreasing of the acf is slow and there is still undesired dependency among our markov chain samples. This dependency explains why we observed an optimized effective size which is quite low for $N = 10^5$ iterations.

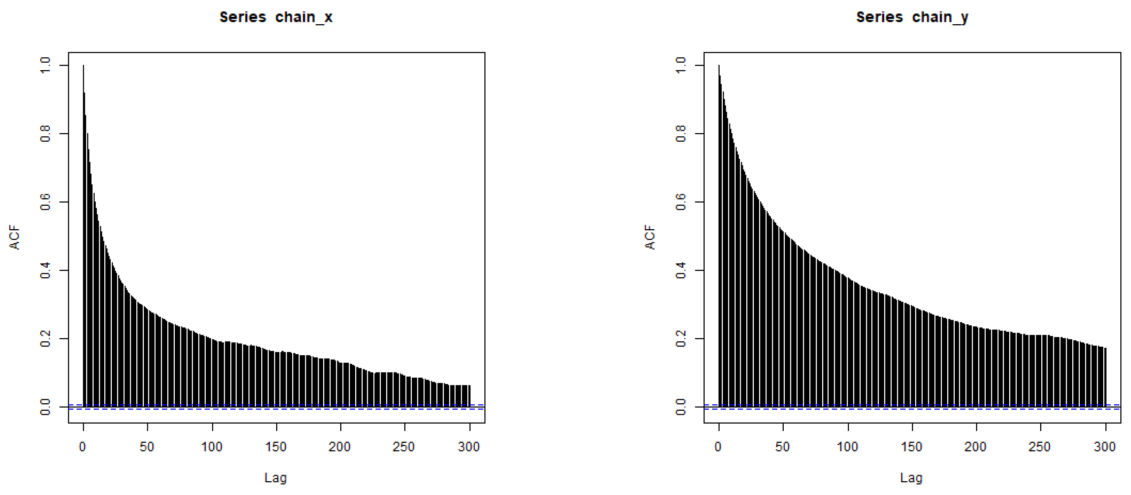

Figure 2: Autocorrelation plots for the chains $\left(X^t\right)_{t=1,..,N}$ and $\left(Y^t\right)_{t=1,..,N}$

(b) The previous Metropolis-Hasting algorithm with $N = 10^5$ iterations and the stated initial values (X_i^0, Y_i^0) (Table 1) gives us samples of the joint distribution $(X_i^t, Y_i^t)_{t=1,..,N}$ for $i = 1, .., 5$. As suggested before, we do a burn-in of the $N_{bi} = 2000$ first elements for each chain. The ergodic theorem gives the following approximation for $(X, Y) \sim f$:

$$\mathbb{E}(X) \simeq \frac{1}{5}\sum_{i=1}^{5}\frac{1}{N-N_{bi}}\sum_{t=N_{bi}+1}^{N} X_i^t \qquad \mathbb{E}(Y) \simeq \frac{1}{5}\sum_{i=1}^{5}\frac{1}{N-N_{bi}}\sum_{t=N_{bi}+1}^{N} Y_i^t$$

We compute the estimates :

$$\mathbb{E}(X) \simeq -0.272 \qquad \mathbb{E}(Y) \simeq 1.387$$

The density f is symmetric with respect to x, then we know that we should obtain $\mathbb{E}(X) = 0$. We can also compute thanks to **R** function **integrate** $\mathbb{E}(X) \simeq 1.633$. The error for our estimates are in the order of 3×10^{-1}.

(c) In the article *Adaptive proposal distribution for random walk Metropolis algorithm* [1], Heikki Haario, Eero Saksman and Johanna Tamminen propose a MCMC algorithm with an adaptive proposal distribution. This means that the proposal distribution varies from one iteration t of the Metropolis-algorithm to another :

$$q_t(.|Z^1, .., Z^t) \sim \mathcal{N}(Z^t, c_d^2 R_t) \tag{5}$$

where $Z^t = (X^t, Y^t)$, R_t is the $d \times d$ covariance matrix $[R_t]_{ij} = [Cov(Z_{t-H+i}, Z_{t-H+j})]_{(i,j)\in\{1,..,H\}}$ and c_d is a scaling factor. The following lets us compute a proposed state $\tilde{Z}^t$ from the current position Z^{t-1} :

$$\tilde{Z}^t \sim Z^{t-1} + \frac{c_d}{\sqrt{H-1}}\tilde{K}^T \mathcal{N}(0, I_H) \tag{6}$$

Where $\tilde{K} \in \mathbb{R}^{H\times 2}$ is the vector $(Z_{t-H+1}, ..., Z_t)$ centered. In practice, we update the covariance of the proposal distribution each U steps. We give the pseudo-code of the Metropolis-Hasting algorithm for the adaptive proposal distributions q_t which we notice to be symmetric 2.

Algorithm 2 Metropolis-Hastings algorithm with adaptive proposal

```
1: Given initial values $(X^0, Y^0)$, the target density $f$ as defined in equation 1, the covariance $2 \times 2$ matrix $\Sigma$, the number or iterations $N$, the coefficients $c_d$, $H$, $U$ and the function *step_metropolis_hasting* that encodes one step of the Metropolis-Hasting algorithm defined previously (Algorithm 1).
2: for t = 1,2,...H do :
3:     $(X^t, Y^t) = step\_metropolis\_hasting((X^{t-1}, Y^{t-1}), f, \Sigma, N = H)$
4: $K = (X_1, ..., X_H)$
5: $\tilde{K} = K - \mathbb{E}(K)$
6: for $t = H+1, .., N$ do :
7:     if: $t$ divides $U$ then:
8:         $K = (X_{t-H+1}, ..., X_t)$
9:         $\tilde{K} = K - \mathbb{E}(K)$
10:    Let $(\tilde{X}, \tilde{Y}) \sim (X^t, Y^t) + \frac{c_d}{\sqrt{H-1}}\tilde{K}^T \mathcal{N}(\mathbf{0}, I_H)$
11:    Let $U \sim \mathcal{U}([0,1])$
12:    if: $U \leq \min\left\{\exp\left[\log(f(\tilde{X},\tilde{Y})) - \log(f(X^{t-1}, Y^{t-1}))\right], 1\right\}$ then:
13:        $(X^t, Y^t) = (\tilde{X}, \tilde{Y})$
14:    else:
15:        $(X^t, Y^t) = (X^{t-1}, Y^{t-1})$
```

As stated by Heikki Haario, Eero Saksman and Johanna Tamminen [1], we choose $H = 200$, $U = 200$ and $c_d = 2.4/\sqrt{2}$ for a proposal in 2-dimensions. We generate samples with adaptive MCMC algorithm for $N = 10^5$ iterations with a burn-in of the first 5000 elements. The traceplot indicates that the chain explore well the full state. When we plot the joint distributions $\left(X^t, Y^t\right)_{t=1,..,N}$ obtained, the adaptive proposal seems to sample similarly than the simple random walk (Figure 4). However, the extreme values of the "banana" shape are more explored with the adaptive proposal. Moreover, the decreasing in the acf plot is much faster with the adaptive proposal than with the simple random walk proposal. The effective size is higher too. For our unimodal target distributions f, the generated chain has approximately stationary distribution f [1]. We compute estimates of the mean of $(X, Y) \sim f$ with ergodic theorem :

$$\mathbb{E}(X) \simeq -0.0329 \qquad \mathbb{E}(Y) \simeq 1.627$$

We notice an improvement of the precision of our estimates of $\mathbb{E}(X)$ and $\mathbb{E}(Y)$. We now obtain errors in the order of 3×10^{-2}. The main advantage of the use of adaptive proposal is that we avoid to tune manually the parameters of the proposal distribution (σ_x, σ_y and ρ in our example). Even with a grid search that required time and important computational ressources, we obtained parameters that lead to an acf slowly decreasing. The adaptive proposal sampled more accurately the target density especially for extreme values. Thus it obtained better results to compute the mean.

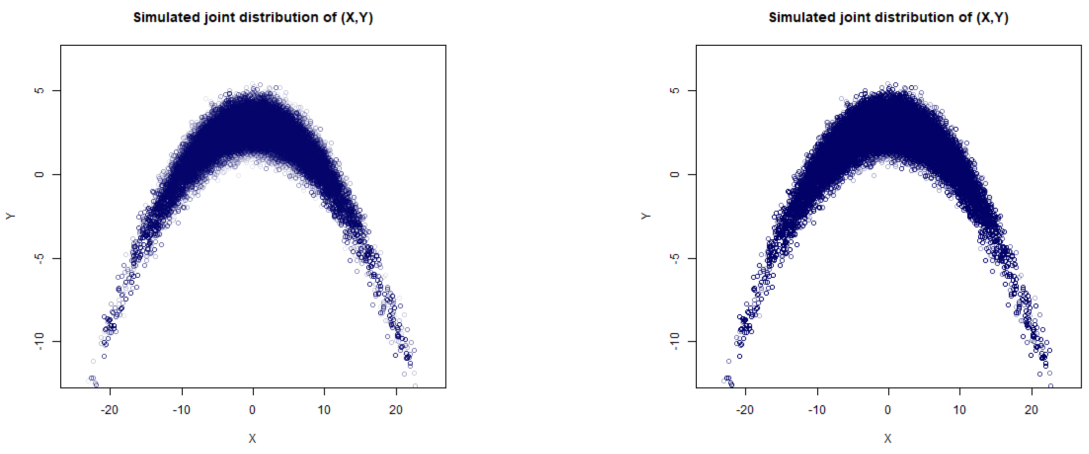

Figure 3: Plots of the sampled joint density $\left(X^t, Y^t\right)_{t=1,..,N}$ with simple MH algorithm (right) and adaptative MH algorithm (left)

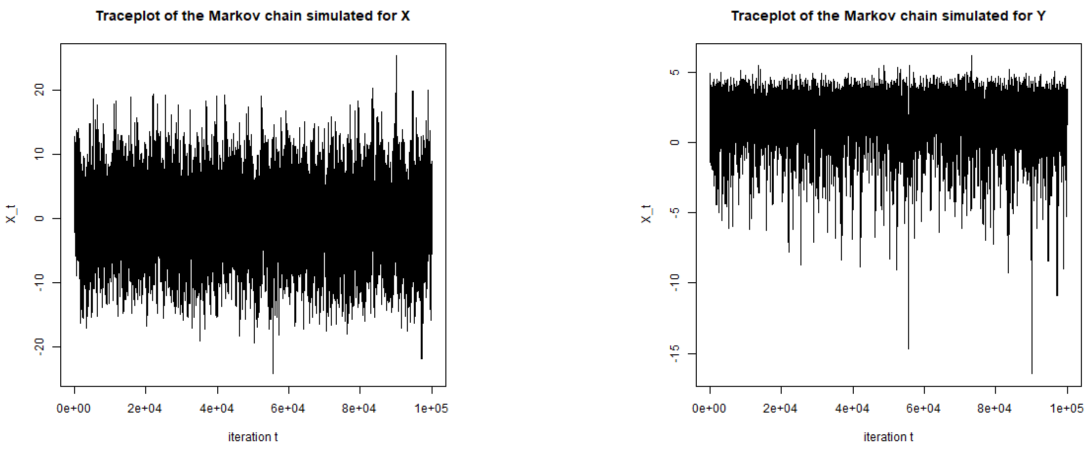

Figure 4: Traceplots of the samples $\left(X^t\right)_{t=1,..,N}$ (right) and $\left(Y^t\right)_{t=1,..,N}$ (left) for different initial values with adaptative MH algorithm

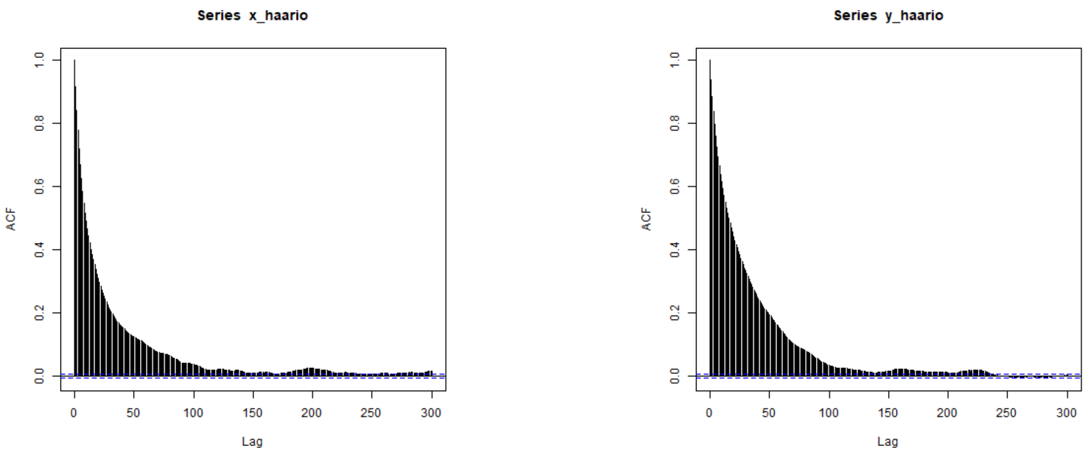

Figure 5: Autocorrelation plots for the chains $\left(X^t\right)_{t=1,..,N}$ (left) and $\left(Y^t\right)_{t=1,..,N}$ (right)

Question 2

Consider the following target density :

$$g(\theta) = k\left(\frac{4}{10}\exp\left\{-(a-\theta)^2\right\} + \frac{6}{10}\exp\left\{-(30-\theta)^2\right\}\right) \tag{7}$$

where $a = 4$ is the last digit of my CID and k is a normalising constant.

(a) We implement a random-walk Metropolis-Hastings algorithm with a proposal :

$$\tilde{\theta}^t \sim \mathcal{N}(\theta^{t-1}, \sigma^2)$$

for some fixed $\sigma > 0$. We tried to run the algorithm with different values of σ and compare the behaviour of the obtained Markov chain realisations.

One one hand, for low values of σ, the chain will only explore one of the two modes of the target density g. We plot the traceplot of Markov chains generated from different initial values $\theta^0 \in \{0, 5, 25, 30\}$. We ran $N = 5000$ iterations of the Metropolis-Hastings sampler with a standard deviation of the proposal equal to $\sigma = 1$. The Markov chains generated are observed to stay in a neighborhood of either the mode $\theta_1^* \simeq a$ or $\theta_2^* \simeq 30$. The gelman coefficient is computed equal to 29.1 with an upper 95% confidence interval equal to 51.3 which also indicates that the chain is not mixing well (Table 2).

On the other hand, for large values of σ, the generated chain will be able to explore the two modes of the target density g. We plot the traceplot of Markov chains generated from different initial values $\theta^0 \in \{0, 5, 25, 30\}$. We ran $N = 5000$ iterations of the Metropolis-Hastings sampler with a standard deviation of the proposal equal to $\sigma = 10$. However, The acceptance rate is equal to 0.09 which indicates that there is a vast waste of simulations. The effective sample size is also noticed to be very low (Table 2). The markov chain generated does not behave as an independent sample and therefore would not be appropriate to estimate properties of the density g.

We observe from these trials that in every case, the sampler does not explore the full space well. We may use Multiple chain MCMC methods to counter this problem.

	ESS ($\theta^0 = 0$)	ESS ($\theta^0 = 5$)	ESS ($\theta^0 = 25$)	ESS ($\theta^0 = 30$)	acceptance rate	gel_1	gel_1_CI
$\sigma = 1$	1699.6	1778.6	1602.3	1649.1	0.61	29.1	51.3
$\sigma = 2$	2281.5	2073.8	2281.0	2404.6	0.40	29.1	51.5
$\sigma = 5$	1293.0	1188.4	1161.1	1187.3	0.17	29.5	52.2
$\sigma = 10$	311.6	335.8	330.9	306.3	0.09	1.1	1.2
$\sigma = 20$	183.0	209.9	206.4	212.2	0.06	1.0	1.0

Table 2: ESS, acceptance rate and gelman coefficient with different initial conditions and different σ

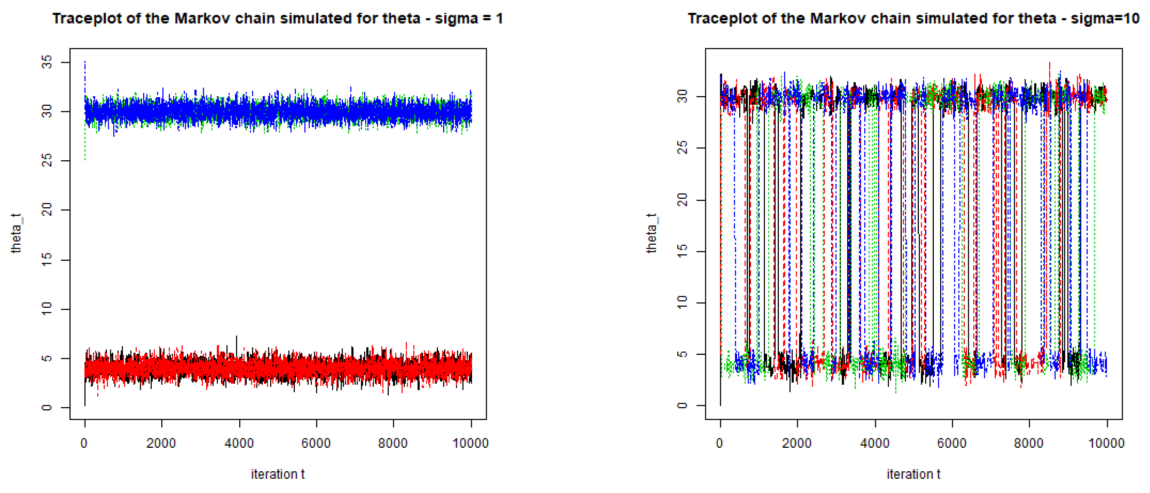

Figure 6: Traceplots of the Markov chains simulated for $\sigma = 1$ (left) and $\sigma = 10$ (right)

(b) We aim to overcome the problems met in the previous question by using a parallel tempering algorithm. First, we write the target density g (Equation 7) under the form :

$$g(\theta) \propto \exp\{-H(\theta)\}$$

where $H(\theta)$ is defined for $\theta \in \mathbb{R}$ by :

$$H(\theta) = -\log\left\{\frac{4}{10}\exp\{-(a-\theta)^2\} + \frac{6}{10}\exp\{-(30-\theta)^2\}\right\} \tag{8}$$

We also define for $m = 1, .., M = 7$ the probability densities :

$$g_m(\theta) \propto \exp\left\{-\frac{H(\theta)}{T_m}\right\} \tag{9}$$

Where $1 = T_1 < T_2 < ... < T_M$. The parallel tempering construct samples of a M-dimension Markov chain with stationary distribution :

$$g^M(\theta) = \prod_{m=1}^{M} g_m(\theta_m)$$

We chose values of T_m for $m = 1, .., M$ such that g_m is relatively “flat” for $m = 5, 6, 7$, and g_m behaves approximately like g_1 for $m = 2, 3$ (See Table 3). As we are in one-dimension, we can use the **R** function **integrate** to compute the constant k and check that the densities g_m for the different values of m (Figure 7) have the desired behaviour stated before. Moreover, the within chain updates in the parallel tempering algorithm needs a proposal distribution for each of the Markov chain $(\theta_1^t, .., \theta_M^t)_{t=1,..,N}$. For $m = 1, .., M$, we propose a random walk proposal $q_m(\theta_m^t, .) \sim \mathcal{N}(\theta_m^{t-1}, \sigma_m^2)$ where the chosen values for σ_m are increasing with m (Table 3). This choice will enable the chain $(\theta_m^t)_{(m,t)\in\{1,..,M\}\times\{1,..,N\}}$ to explore well the support of the density g_m. For $m = 1, .., M$, the transition kernel P_m of the chain $(\theta_m^t)_{t=1,..,N}$ is then given for $\theta \in \mathbb{R}$ by :

$$P_m(\theta_m^{t-1}, \theta) = q_m(\theta_m^{t-1}, \theta) \times \min\left(\frac{g_m(\theta)}{g_m(\theta_m^{t-1})}, 1\right)$$

We will check a posteriori that these chosen values propose a good trade-off between exploration and acceptance rate.

m	1	2	3	4	5	6	7
T_m	1	2	4	5	10	20	50
σ_m	0.25	0.5	1	2	5	10	15

Table 3: Temperature values and associated standard deviation for the proposal distributions $g_1, .., g_M$

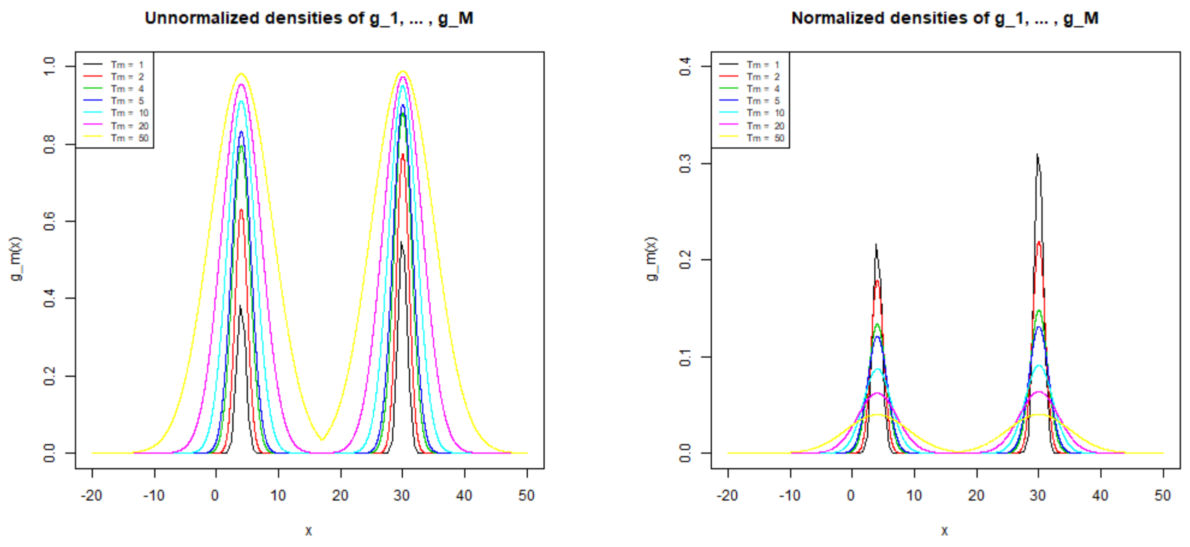

Figure 7: Unnormalized and normalized densities $g_1, .., g_M$

Then, we introduce the function q_B defined for $(l,m) \in \{1,2,..,M\} \times \{1,2,..,M\}$ by :

$$q_B(l,m) = \left| \begin{array}{l} \frac{1}{2} \text{ if } 2 \leq l \leq M-1 \text{ and } m \in \{l-1, l+1\} \\ 1 \text{ if } l=1 \text{ and } m=2, \text{ or } l=M \text{ and } m=M-1 \\ 0 \text{ otherwise} \end{array} \right.$$

We can now compute a parallel tempering that sample from the joint distribution g^M. We give the pseudo-code of the parallel tempering algorithm for the adaptive proposal distributions q_t which we notice to be symmetric 3.

Algorithm 3 Parallel tempering algorithm

```
1:  Given initial values $(\theta_1^0, .., \theta_M^0)$, the target densities $g_1, .., g_M$ as defined in equation 9, the standard deviations $\sigma_1, .., \sigma_M$ for the random proposals (Table 3), the function $q_B$, the number of iterations $N$
2:  for $t = 1, 2, .., N$ do :
3:    for $m = 1, 2, .., M$ do :
4:      $\tilde{\theta}_m^t \sim \mathcal{N}(\theta_m^{t-1}, \sigma_m^2)$
5:      $U \sim U([0,1])$
6:      if: $U \leq \min\left(\frac{g_m(\tilde{\theta}_m^t)}{g_m(\theta_m^{t-1})}, 1\right)$ then:
7:        $\theta_m^t = \tilde{\theta}_m^t$
8:      else:
9:        $\theta_m^t = \theta_m^{t-1}$
10:   for $p = 1, 2, .., M$ do :
11:     $l \sim \mathcal{U}(\{1, 2, .., M\})$
12:     if: $l == 1$ then: $k = 1$
13:     elif: $l == M$ then: $k = M - 1$
14:     else:
15:       $V \sim \mathcal{U}([0,1])$
16:       if: $V < 1/2$ then: $k = l - 1$
17:       else: $k = l + 1$
18:     $W \sim \mathcal{U}([0,1])$
19:     if: $W < \min\left(\frac{g_l(\theta_k^t) g_k(\theta_l^t) q_B(k,l)}{g_l(\theta_l^t) g_k(\theta_k^t) q_B(l,k)}, 1\right)$ then:
20:       $(\theta_l^t, \theta_k^t) = (\theta_k^t, \theta_l^t)$
```

By construction, the marginal distributions are $g_1, .., g_M$ and the sample obtained $(\theta_1^t)_{t=1,..,N}$ has stationary distribution $g_1 = g$. We construct chains for different initial values and the gelman coefficient is equal to 1 as well as its confidence interval upper value. Our constructed chain is mixing well. We simulate the chain $(\theta_1^t)_{t=1,..,N}$ sampled for $N = 10^5$ iterations with initial value whose all component are equal to zero (Figure 8). Its traceplot indicates that the chain seems to explore well the full space. Especially, it explores both the neighboorhood of the two modes (located around $a = 4$ and 30) of the target distribution. Moreover, the acf plot is decreasing quite quickly. A density plot constructed from the histogram of our chain shows that we are correctly sampling from the target distribution (Figure 9).

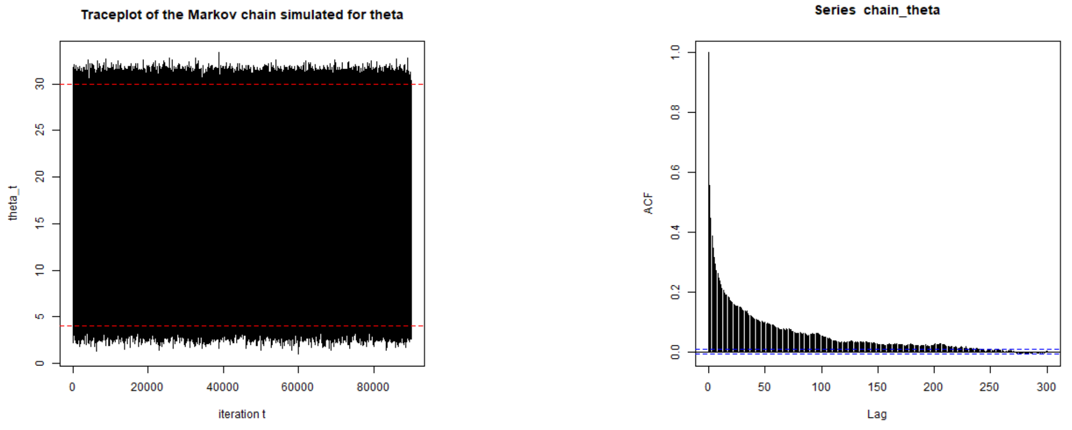

Figure 8: Traceplot and autocorrelation plot of the sample generated with parallel tempering

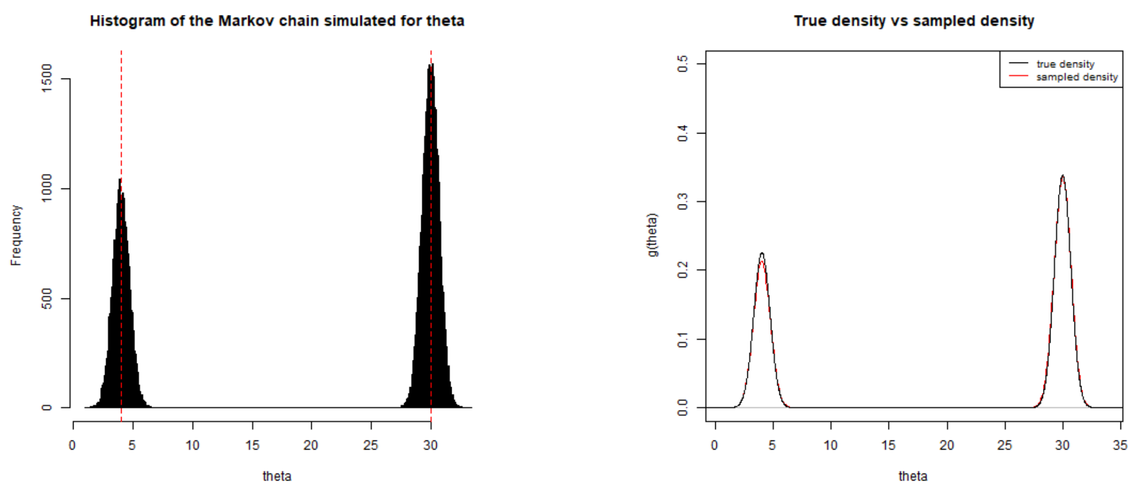

Figure 9: Histogram and density plot of the sample generated with parallel tempering

(c) We want to compute $\mathbb{P}(\theta > 20)$. As stated before, the parallel tempering gives realisations $\left(\theta_1^t\right)_{t=1,..,N}$ of a Markov chain with stationary distribution g. We generate $N = 10^5$ samples and we do a burn-in of the $N_{bi} = 5000$ first elements. By the ergodic theorem, we compute the estimate of $\mathbb{P}(\theta > 20)$ for $\theta \sim g$:

$$\mathbb{P}(\theta > 20) \simeq \frac{1}{N - N_{bi}} \sum_{t=N_{bi}}^{N} 1_{(20,+\infty)}(\theta_1^t)$$

Therefore, we compute :

$$\mathbb{P}(\theta > 20) \simeq 0.6051$$

To check this result, we compare it to the result obtained using numerical integration. Recall the definition of the quantity of interest :

$$\mathbb{P}(\theta > 20) = \int_{20}^{\infty} g(\theta) \mathrm{d}\theta$$

To calculate this integral, we need to compute the normalizing constant k that is given by :

$$k^{-1} = \int_{-\infty}^{\infty} \left(\frac{4}{10} \exp\{-(a-\theta)^2\} + \frac{6}{10} \exp\{-(30-\theta)^2\} \right) \mathrm{d}\theta \tag{10}$$

Then, **R** function **integrate** gives $k \simeq 0.564$, and it is then computed :

$$\mathbb{P}(\theta > 20) \simeq 0.6000$$

The results are close with an error of 5×10^{-3}, the method seems to work well. We can improve the precision by choosing more precisely the values of the temperatures $(T_m)_{m=1,..,M}$ and the standard deviations $(\sigma_m)_{m=1,..,M}$ of the associated proposal distributions. One could also choose to increase the number of observations generated with the Metropolis-Hastings sampler.

Appendices

A R code - Question 1

```
###   MATH97125 - Computational Statistics , Coursework final , question 1 ###

### code of Robin Mathelier ###


setwd("C:/Users/robin/Dropbox/Applications/Overleaf/Coursework_Final_Computational")
personnaliser
getwd()
.libPaths("C:/Users/robin/Documents/R/win-library/3.6")
rm(list=objects())

library(xtable)
library(latex2exp)
library(expm)
library(coda)
library(ggplot2)
library(MASS)
library('pracma')

set.seed(42)

## Question 1 ##

f = function(z) exp(-z[1]^2/100 - (z[2]+3/100*z[1]^2 -3)^2)

log_f = function(z) -z[1]^2/100 - (z[2]+3/100*z[1]^2 -3)^2

f = function(x,y) exp(-x^2/100 - (y+3/100*x^2 -3)^2)
IntY <- function(x) sapply(x, function(b) integrate(function(y) f(b,y),-Inf,Inf)$valu
integrate(function(x) x*IntY(x), -Inf, Inf)$value
IntX <- function(y) sapply(y, function(b) integrate(function(x) f(x,b),-Inf,Inf)$valu
k = integrate(function(y) IntX(y), -Inf, Inf)$value
integrate(function(y) y*IntX(y), -Inf, Inf)$value/k

mvdnorm = function(vec,mu,corr_matrix) {
  x=vec[1]
  y=vec[2]
  x_mu = as.matrix(c(x,y)-mu)
  return( (1/(2*pi*sqrt(det(corr_matrix))))*
  exp((-1/2)*t(x_mu)%*%solve(corr_matrix)%*%x_mu) )
}

create_chains = function(init , corr_matrix){
  q <- function(vec_1,vec_2) mvdnorm(vec_1,mu=vec_2,corr_matrix=corr_matrix)
  MHpost <- function(n=1e4,init){
     xall <- matrix(NA,ncol=n, nrow=2)
     x <- init
     xall[,1] <- x
     for (t in seq(2,n)){
        y <- mvrnorm(1,mu=x,Sigma=corr_matrix)
        if (runif(1)<=(exp(log_f(y)-log_f(x))*q(y,x)/q(x,y))){
           x<-y
        }
        xall[,t] <- x
      }
      return(xall)
    }
  return(MHpost(init=init))
}

theta = c(0,3)

create_corr_mat = function(sx,sy,ro) rbind(c(sx^2,ro*sx*sy),
                                            c(ro*sx*sy,sy^2))

corr_matrix = create_corr_mat(sx=5,sy=1,ro=0.5)

chains = create_chains(init=theta,corr_matrix=corr_matrix)
dim(chains)

chain_x = chains[1,]
traceplot(mcmc(chain_x))

chain_y = chains[2,]
traceplot(mcmc(chain_y))

effectiveSize(chain_x)
effectiveSize(chain_y)

acf(chain_x,lag.max=100)
acf(chain_y)

list_sx = linspace(7,9,n=4)
list_sy = linspace(1,2,n=4)
list_ro = c(0.005,0.01,0.05,0.1,0.15)

set.seed(42)

ESS_x = 0
ESS_y = 0
for (sx in list_sx){
  for (sy in list_sy){
    for (ro in list_ro){
      corr_matrix = create_corr_mat(sx,sy,ro)
      if(eigen(corr_matrix)$values[1]>0 & eigen(corr_matrix)$values[2]>0){
        chains = create_chains(init=theta,corr_matrix=corr_matrix)
        chain_x = chains[1,]
        chain_y = chains[2,]
        new_ESS_x = effectiveSize(mcmc(chain_x))
        new_ESS_y = effectiveSize(mcmc(chain_y))
        if (new_ESS_x > ESS_x & new_ESS_y > ESS_y){
          ESS_x = new_ESS_x
          ESS_y = new_ESS_y
          sx_chosen = sx
          sy_chosen = sy
          ro_chosen = ro
          chain_x_chosen = chain_x
          chain_y_chosen = chain_y
        }
```

```
            }
        }
    }
}


cat('sx:',sx_chosen,'sy:',sy_chosen,'rho',ro_chosen)
chain_x = chain_x_chosen
chain_y = chain_y_chosen

acf(chain_x,lag.max=100)
acf(chain_y,lag.max=100)

effectiveSize((chain_x))
effectiveSize((chain_y))

traceplot(mcmc(chain_x))
traceplot(mcmc(chain_y))

plot(chain_x,chain_y)

chain_x_ts = ts(chain_x)
chain_y_ts = ts(chain_y)

acf(ts.union(chain_x_ts,chain_y_ts))

create_chains_def = function(init, corr_matrix,n_iter=1e4){
  q <- function(vec_1,vec_2) mvdnorm(vec_1,mu=vec_2,corr_matrix=corr_matrix)
  MHpost <- function(n=n_iter,init){
    xall <- matrix(NA,ncol=n, nrow=2)
    x <- init
    xall[,1] <- x
    for (t in seq(2,n)){
      y <- mvrnorm(1,mu=x,Sigma=corr_matrix)
      if (runif(1)<=(exp(log_f(y)-log_f(x)))){
        x<-y
      }
      xall[,t] <- x
    }
    return(xall)
  }
  return(MHpost(init=init))
}

set.seed(42)
theta=c(0,3)

corr_matrix = create_corr_mat(sx=sx_chosen,sy=sy_chosen,ro=ro_chosen)
chains = create_chains_def(init=theta,corr_matrix,n_iter=1e5)

chain_x = chains[1,]
chain_y = chains[2,]

acf(chain_x,lag.max=300)
acf(chain_y,lag.max=300)

png(filename='acf_x.png')
```

```
acf(chain_x, lag.max=300)
dev.off()

png(filename='acf_y.png')
acf(chain_y, lag.max=300)
dev.off()

effectiveSize((chain_x))
effectiveSize((chain_y))

plot(chain_x, chain_y,
     col=rgb(red=0,green=0,blue=0.4,alpha=0.1),
     xlim=c(-25,25)
     ,xlab = 'X',ylab='Y',main = 'Simulated joint distribution of (X,Y)')

png(filename='plot_x_y.png')
plot(chain_x, chain_y,
     col=rgb(red=0,green=0,blue=0.4,alpha=0.1),
     xlim=c(-25,25),ylim=c(-12,7),
     xlab = 'X',ylab='Y',main = 'Simulated joint distribution of (X,Y)')
dev.off()

traceplot(mcmc(chain_x))
traceplot(mcmc(chain_y))

list_theta = list(cbind(-20,-20),
                  cbind(-10,-10),
                  cbind(0,0),
                  cbind(10,10),
                  cbind(20,20))

set.seed(42)

chains_x=c()
chains_y=c()

for (i in 1:length(list_theta)){
  theta = as.vector(list_theta[[i]])
  chain = create_chains_def(init=theta, corr_matrix, n_iter = 5e4)
  chain_x=chain[1,]
  chain_y=chain[2,]
  chains_x=rbind(chains_x, chain_x)
  chains_y=rbind(chains_y, chain_y)
}

set.seed(42)

chains_x_mcmc <- lapply(1:length(list_theta),
                        function(i) mcmc(chains_x[i,]))

chains_y_mcmc <- lapply(1:length(list_theta),
                        function(i) mcmc(chains_y[i,]))

png(filename='traceplot_x_init_values.png')
traceplot(chains_x_mcmc, main='Traceplot of the Markov chain simulated for X',
          ylab = 'X_t', xlab = 'iteration t')
abline(v=2000,lwd=3,lty=2)
```

```
dev.off()

png(filename='traceplot_y_init_values.png')
traceplot(chains_y_mcmc,main='Traceplot of the Markov chain simulated for Y',
          ylab = 'Y_t',xlab = 'iteration t')
abline(v=2000,lwd=3,lty=2)
dev.off()

gelman.diag(chains_x_mcmc)
gelman.diag(chains_y_mcmc)

### Question 2

chains_x_thin = sapply(seq(5), function(i) mean(chains_x[i,seq(1000,length(chains_x[i
mean(chains_x_thin)

means_mc_x <- sapply(seq(5), function(i) mean(chains_x[i,-seq(2000)]))
mean(means_mc_x)

means_mc_y <- sapply(seq(5), function(i) mean(chains_y[i,-seq(2000)]))
mean(means_mc_y)

chains_x_bi = chains_x[,-seq(2000)]
b = 100
n = length(chains_x_bi)
mub_x = diff(cumsum(chains_x_bi)[(0:(n/b))*b])/b
acf(mub_x)
mean(mub_x)

chains_y_bi = chains_y[,-seq(2000)]
b = 100
n = length(chains_y_bi)
mub_y = diff(cumsum(chains_y_bi)[(0:(n/b))*b])/b
acf(mub_y)
mean(mub_y)

### Question 3

d=2
H=200
c=2.4/sqrt(2)


create_chains_haario = function(init,
                                H=200,U=200,
                                c=2.4/sqrt(2),
                                n_iter=1e4,
                                corr_matrix=diag(c(1,1))){
  MHpost <- function(init){
    xall <- matrix(NA, ncol=n_iter, nrow=2)
    x <- init
    xall[,1] <- x
    for (t in seq(2,H)){
      y <- mvrnorm(1,mu=x,Sigma=corr_matrix)
      if (runif(1)<=(exp(log_f(y)-log_f(x)))){
        x<-y
      }
      xall[,t] <- x
    }
    return(xall)}
    x=init
    xall = MHpost(init=init)
   K = t(xall[,1:H])
   exp_K = sapply(1:2,function(j) mean(K[,j]))
   tilde_K = sapply(1:2,function(j) K[,j]-exp_K[j])
   for(t in (H+1):n_iter){
      if(t%%U==0){
        K = t(xall[,(t-H):(t-1)])
        exp_K = sapply(1:2,function(j) mean(K[,j]))
        tilde_K = sapply(1:2,function(j) K[,j]-exp_K[j])
      }
      y <- x+(c/sqrt(H-1))*t(tilde_K)%*%cbind(rnorm(H))
      if (runif(1)<=(exp(log_f(y)-log_f(x)))){x<-y}
      xall[,t] <- x
      }
  return(xall)
    }

set.seed(44)

chains=create_chains_haario(init=c(0,3),n_iter=1e5)

x_haario = chains[1,]
y_haario = chains[2,]

png(filename='traceplot_x_haario.png')
traceplot(mcmc(x_haario),main='Traceplot of the Markov chain simulated for X',
          ylab = 'X_t',xlab = 'iteration t')
dev.off()

png(filename='traceplot_y_haario.png')
traceplot(mcmc(y_haario),main='Traceplot of the Markov chain simulated for Y',
          ylab = 'X_t',xlab = 'iteration t')
dev.off()

plot(x_haario,y_haario,
     col=rgb(red=0,green=0,blue=0.4,alpha=0.3),
     xlim=c(-25,25)
     ,xlab = 'X',ylab='Y',main = 'Simulated joint distribution of (X,Y)')

png(filename='plot_x_y_haario.png')
plot(x_haario,y_haario,
     col=rgb(red=0,green=0,blue=0.4,alpha=0.3),
     xlim=c(-25,25),ylim=c(-12,7),
     xlab = 'X',ylab='Y',main = 'Simulated joint distribution of (X,Y)')
dev.off()

acf(x_haario,lag.max=300)
acf(y_haario,lag.max=300)

png(filename='acf_x_haario.png')
acf(x_haario,lag.max=300)
dev.off()
```

```
png(filename='acf_y_haario.png')
acf(y_haario,lag.max=300)
dev.off()

x_haario_ts = ts(x_haario)
y_haario_ts = ts(y_haario)

acf(ts.union(x_haario_ts,y_haario_ts))

x_haario_thin = x_haario[seq(500,length(x_haario),10)]
mean(x_haario_thin)

y_haario_thin = y_haario[seq(500,length(x_haario),10)]
mean(y_haario_thin)

mean(x_haario[-seq(1,5000)])
mean(y_haario[-seq(1,5000)])

effectiveSize(x_haario)
effectiveSize(y_haario)
```

B R code - Question 2

```
###  MATH97125 - Computational Statistics, Coursework final, question 2 ###

### code of Robin Mathelier ###

setwd("C:/Users/robin/Dropbox/Applications/Overleaf/Coursework_Final_Computational")
personnaliser
getwd()
.libPaths("C:/Users/robin/Documents/R/win-library/3.6")
rm(list=objects())


library(xtable)
library(latex2exp)
library(expm)
library(coda)
library(ggplot2)
library(MASS)
library(pracma)
library('purrr')


set.seed(42)

## part a

a = 4

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

plot(g,from=0,to=40,main='plot of g(x)',xlab='x',ylab='g(x)')

mh_rw = function(n=1e4,sigma=1,init=0){
  Xres = rep(NA,n)
  X = init
  for (i in 1:n){
    Y = X+rnorm(1,mean=0,sd=sigma)
    if (runif(1) <= g(Y)/g(X)){X = Y}
    Xres[i] = X
  }
  Xres
}

chain = mh_rw()

inits = c(0,5,25,35) ## starting values
inits

# sigma = 1

ess_s1 = lapply(1:length(inits),
               function(i) effectiveSize(mh_rw(init=inits[i], sigma=1)))
ess_s1

chains_1 = lapply(1:length(inits),
                 function(i) mcmc(mh_rw(init=inits[i], sigma=1)))
chains_1 = mcmc.list(chains_1)
gel_1 = gelman.diag(chains_1)[[1]][1]
gel_1_CI = gelman.diag(chains_1)[[1]][2]

chain_1 = mh_rw(init=inits[1], sigma=1)
n_diff_1 = 1
for (i in 1:(length(chain_1)-1)){
  if(chain_1[i+1] != chain_1[i]){n_diff_1= n_diff_1+1}
}
accept_rate_1 = n_diff_1/length(chain_1)
accept_rate_1

chains = lapply(1:length(inits), function(i) mcmc(mh_rw(init=inits[i], sigma=1)))
png(filename='traceplot_s1.png')
traceplot(chains,main='Traceplot␣of␣the␣Markov␣chain␣simulated␣for␣theta␣-␣sigma␣=␣1
          ylab='theta_t',xlab='iteration␣t')
dev.off()

# sigma = 2

ess_s2 = lapply(1:length(inits),
               function(i) effectiveSize(mh_rw(init=inits[i], sigma=2)))
ess_s2

chains_2 = lapply(1:length(inits),
                 function(i) mcmc(mh_rw(init=inits[i], sigma=2)))
chains_2 = mcmc.list(chains)
gel_2 = gelman.diag(chains_2)[[1]][1]
gel_2_CI = gelman.diag(chains_2)[[1]][2]

chain_2 = mh_rw(init=inits[1], sigma=2)
n_diff_2 = 1
for (i in 1:(length(chain_2)-1)){
  if(chain_2[i+1] != chain_2[i]){n_diff_2= n_diff_2+1}
}
```

```
accept_rate_2 = n_diff_2/length(chain_2)
accept_rate_2

# sigma = 5

ess_s3 = lapply(1:length(inits),
                function(i) effectiveSize(mh_rw(init=inits[i], sigma=5)))
ess_s3

chains_3 = lapply(1:length(inits),
                  function(i) mcmc(mh_rw(init=inits[i], sigma=5)))
chains_3 = mcmc.list(chains_3)
gel_3 = gelman.diag(chains_3)[[1]][1]
gel_3_CI = gelman.diag(chains_3)[[1]][2]

chain_3 = mh_rw(init=inits[1], sigma=5)
traceplot(mcmc(chain_3))
n_diff_3 = 1
for (i in 1:(length(chain_3)-1)){
  if(chain_3[i+1] != chain_3[i]){n_diff_3= n_diff_3+1}
}
accept_rate_3 = n_diff_3/length(chain_3)
accept_rate_3

# sigma = 10

ess_s4 = lapply(1:length(inits),
                function(i) effectiveSize(mh_rw(n=1e5,init=inits[i], sigma=10)))
ess_s4

chains_4 = lapply(1:length(inits),
                  function(i) mcmc(mh_rw(init=inits[i], sigma=10)))
chains_4 = mcmc.list(chains_4)
gel_4 = gelman.diag(chains_4)[[1]][1]
gel_4_CI = gelman.diag(chains_4)[[1]][2]

chain_4 = mh_rw(init=inits[1], sigma=10)
n_diff_4 = 1
for (i in 1:(length(chain_4)-1)){
  if(chain_4[i+1] != chain_4[i]){n_diff_4= n_diff_4+1}
}
accept_rate_4 = n_diff_4/length(chain_4)
accept_rate_4

chains = lapply(1:length(inits), function(i) mcmc(mh_rw(init=inits[i], sigma=10)))
png(filename = 'traceplot_s2.png')
traceplot(chains,main='Traceplot␣of␣the␣Markov␣chain␣simulated␣for␣theta␣-␣sigma=10'
          ylab='theta_t',xlab='iteration_t')
dev.off()

# sigma = 20

ess_s5 = lapply(1:length(inits),
                function(i) effectiveSize(mh_rw(init=inits[i], sigma=20)))
ess_s5

chains_5 = lapply(1:length(inits),
                    function(i) mcmc(mh_rw(init=inits[i], sigma=20)))
chains_5 = mcmc.list(chains_5)
gel_5 = gelman.diag(chains_5)[[1]][1]
gel_5_CI = gelman.diag(chains_5)[[1]][2]

chain_5 = mh_rw(init=inits[1], sigma=20)
n_diff_5 = 1
for (i in 1:(length(chain_5)-1)){
  if(chain_5[i+1] != chain_5[i]){n_diff_5= n_diff_5+1}
}
accept_rate_5 = n_diff_5/length(chain_5)
accept_rate_5


# table

xtable(data.frame(rbind(cbind(t(ess_s1),accept_rate_1,gel_1,gel_1_CI),
                        cbind(t(ess_s2),accept_rate_2,gel_2,gel_2_CI),
                        cbind(t(ess_s3),accept_rate_3,gel_3,gel_3_CI),
                        cbind(t(ess_s4),accept_rate_4,gel_4,gel_4_CI),
                        cbind(t(ess_s5),accept_rate_5,gel_5,gel_5_CI))),
       digits=1)


## part b


T = c(1,2,4,5,10,20,50)
M=length(T)
h = function(theta) -log(g(theta))
g_m = function(theta,m) exp(-h(theta)/T[m])
k = 1/integrate(g,lower=-10,upper=50)$value
g_norm = function(x) k*g(x)

abs = seq(-20,50,0.01)
ord = g_m(abs,m=1)

png(filename='unnormalized_gm.png')
plot(g,xlim=c(-20,50),ylim=c(0,1)
     ,main='Unnormalized densities of g_1, ..... , g_M',
     ylab = 'g_m(x)')

for (i in 2:M){
  ord = g_m(abs,m=i)
  lines(abs,ord,col=i)
}

legend('topleft',paste('Tm = ',T),col=1:M,lwd=1,cex=0.7)
dev.off()

png(filename='normalized_gm.png')
plot(g_norm,xlim=c(-20,50),ylim=c(0,0.4)
     ,main='Normalized densities of g_1, ..... , g_M',
     ylab = 'g_m(x)')

for (i in 2:M){
  k_i = 1/integrate(function(x) g_m(x,m=i),lower=-10,upper=50)$value
  ord = g_m(abs,m=i)*k_i
  lines(abs,ord,col=i)
}

legend('topleft',paste('Tm = ',T),col=1:M,lwd=1,cex=0.7)
dev.off()

q_b = function(l,m){
  if(l==1 & m==2){return(1)}
  if(l==M & m==M-1){return(1)}
  if(m==l-1 || m==l+1){return(0.5)}
  else{return(0)}
}

list_sigma = c(0.25,0.5,1,2,5,10,15)

xtable(data.frame(rbind(1:M,T,list_sigma)))

n=1e5

parallel_tempering = function(n_iter=n,init,liste_sigma=list_sigma){
  Xres = matrix(NA,ncol=n_iter,nrow=M)
  X = cbind(init)
  Xres[,1] = X
  for(t in 2:n_iter){
    X = Xres[,t-1]
    for(m in 1:M){
      Y_m = X[m]+rnorm(1,mean=0,sd=liste_sigma[m])
      if (runif(1) <= min((g_m(Y_m,m=m)/g_m(X[m],m=m)),1)) {X[m] = Y_m}
      Xres[m,t] = X[m]
    }
    for (p in 1:M){
    l = rdunif(1,a=1,b=M)
    if(l==1){m=2}
    if(l==M){m=M-1}
    if(l!= 1 & l!=M){
      u=runif(1)
      if(u<0.5){m=l+1}
      else{m=l-1}
    }
    v = runif(1)
    rapp = exp((h(Xres[l,t])-h(Xres[m,t])) * (1/T[l] - 1/T[m]))
    if (v < min(1,rapp*q_b(m,l)/q_b(l,m))){
      tilde_x = Xres[m,t]
      Xres[m,t] = Xres[l,t]
      Xres[l,t] = tilde_x
    }
    }
  }
  return(Xres)}

inits = rbind(rep(0,M),
              rep(10,M),
              rep(20,M),
              rep(30,M))

set.seed(42)
```

```
chains = sapply(1:dim(inits)[1], function(i) parallel_tempering(init=inits[i,])[1,])

chains_mcmc = lapply(1:dim(inits)[1], function(i) mcmc(chains[,i]))

chains_mcmc_list = mcmc.list(chains_mcmc)
gelman.diag(chains_mcmc_list)

init = rep(0,M)
chains = parallel_tempering(init=init,n_iter = 1e5)

chain_theta = chains[1,-seq(1,5000)]
png(filename='traceplot_theta.png')
traceplot(mcmc(chain_theta),main='Traceplot of the Markov chain simulated for theta'
          ylab='theta_t',xlab='iteration t')
abline(h=30,col='red',lty=2)
abline(h=a,col='red',lty=2)
dev.off()

effectiveSize(chain_theta)

png(filename='acf_theta.png')
acf(chain_theta,lag.max=300)
dev.off()

n_diff = 1
for (i in 1:(length(chain_theta)-1)){
  if(chain_theta[i+1] != chain_theta[i]){n_diff= n_diff+1}
}
accept_rate = n_diff/length(chain_theta)
accept_rate

png(filename = 'histogram_theta.png')
hist(chain_theta,breaks=1000,
     xlab='theta',
     main='Histogram of the Markov chain simulated for theta')
abline(v=a,lty=2,col='red')
abline(v=30,lty=2,col='red')
dev.off()

png(filename = 'density_plot.png')
plot(density(chain_theta,adjust=0.15),ylim=c(0,0.5),
     col='red',xlab='theta',ylab='g(theta)',
     main='True density vs sampled density')
ord = k*g(abs)
lines(abs,ord)
legend('topright',col=c(1,2),lwd=1,cex=0.8,legend=c('true density','sampled density')
dev.off()

mean(chain_theta>20)

## Question 3


k = 1/integrate(g,lower=-10,upper=50)$value
k*integrate(g,lower=10,upper=60)$value
```

References

[1] Heikki Haario, Eero Saksman, and Johanna Tamminen. Adaptive proposal distribution for random walk metropolis algorithm. *Computational statistics*, 14(3):375–395, 1999.