Generating random samples & Markov chain Monte Carlo

All the sampling methods used in statistics are due to Physists. Hence, they require adjustments before applying those to the statistical problems. Although, MCMC commonly referred to as a Bayesian tool for inference. That's an imperfect statement. The main statistical usage of sampling is to do with performing complicated integral.

The main idea: To approximate $\int t(x)P(x \mid \theta)dx$ for some function of x, we can draw samples of x from a distribution $P(x \mid \theta)$. Let $x_1, \ldots, x_n$ are n samples of x from $P(x \mid \theta)$. Then for n large enough, $\int t(x)P(x \mid \boldsymbol{\theta})dx \approx \frac{1}{n}\sum_{i=1}^{n} t(x_i)$. For example, $P(x \mid \theta)$ can be a normal distribution where θ stands for the mean and variance parameters. Hence to compute numerical expectation, we can generate samples and take the sample mean.

For simulation: To examine performance of a model or a mechanism, we often rely on simulations. In order to run a simulation, we need to generate data under some well-specified data generation scheme. Data generation schemes are usually probability models.

For standard univariate distributions, the computer uses the first method to generate random numbers from that distribution. It is easy to generate from uniform.

1 Univariate case

The inverse CDF (ICDF) method: This is only applicable for univariate method.And this is also an immediate application of what we have just seen above in the first example under "Learn a distribution by sampling". Let $F(x)$ is the cumulative distribution function (CDF) of a random variable, X i.e. $F(x) = P(X \leq x)$ and we want to draw samples of X. We know that $F(X) \sim \text{Unif}(0,1)$. (Why? $P(F(X) \leq u) = P(X \leq F^{-1}(u)) = F(F^{-1}(u)) = u$. Thus $F(X) \sim \text{Unif}(0,1)$.) So we draw samples from uniform distribution which will be samples of $F(X)$ and then do F^{-1} transformation on these samples and get samples of X. All the standard packages use this method to generate random variables.

General list of distributions in R can be found here `https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Distributions.html`. In each the link you will find following funtions:

- `d-` Like `dbeta,dnorm,dgamma,...` compute the density value at a given data point i.e. $f(x) = F'(x)$

- p- Like pbeta,pnorm,pgamma,... compute the cumulative distribution function value at a given data point i.e. $F(x) = P(X \leq x)$.

- q- Like qbeta,qnorm,qgamma,... compute the quantile value at a given probability value i.e. $F^{-1}(x)$

- r- Like rbeta,rnorm,rgamma,... generate a random number following the given distribution.

To test ICDF approach, try the following code:

```
seed <- 1000 #Set any seed-value here x-x1 will always be small.

set.seed(seed)

x <- rnorm(1)

set.seed(seed)

y <- runif(1)
x1 <- qnorm(y) #The F^{-1} function

x-x1 #The error is just the numerical inversion error
```

A more elaborate example

```
betasam1 <- rbeta(10000, 8, 10)

plot(density(betasam1), col=1, type = "l") #ploting empirical (empirical means
                                           #samples)  density using samples for Beta(8,10)
                                           #where samples are generated using R function rbeta

#Using ICDF
unifsam <- runif(10000)
betasam2 <- qbeta(unifsam, 8, 10)

points(density(betasam2), col=2, type = "l") #ploting empirical density using samples
                                             # for Beta(8,10) where samples are generated
                                             # using ICDF method

thetagrid <- (1:10000)/10000
points(thetagrid, dbeta(thetagrid, 8, 10), col=3, type = "l") #actual density
```

The following two methods (Accept-reject and importance sampling) have a common thread. To sample from some target distribution $p(\theta)$, we generate samples from a $q(\theta)$ instead and 'select' only a subset. This 'selection step' is designed in a way that the empirical cumulative distribution function of the selected subset converges to the true CDF induced by $p(\theta)$. The choice of $q(\theta)$ would be such that the sampling from $q(\theta)$ is simple and standard packages are available. The next trick lies in designing the selection step. This is the step where the two methods differ.

Accept-reject: The previous method requires one to know, F^{-1} which is a very strong requirement. For any nonstandard distribution, this will not be known anyway. Accept/reject is one way to sample in such scenarios. It can be implemented using R package `AR`. Here we see an example for generating from log-normal distribution. If $\log(x) \sim \text{Normal}(\mu, \sigma^2)$, we say $x \sim$Log-normal(μ, σ^2).

The basic algorithm: In the accept–reject (AR) algorithm, one first samples from an instrumental distribution $q(\theta)$. In a second step, some of the sampled values are rejected to end up with a sample from $p(\theta)$. The distribution $q(\theta)$ is called the proposal distribution and $p(\theta)$, in this context, the target distribution. The AR algorithm assumes that $p(\theta)$ is bounded above by a multiple of $q(\theta)$, i.e. there is a constant $A < \infty$ such that $p(\theta) < Aq(\theta)$ for all θ. Therefore, the distribution q is also called the envelope distribution, A is the envelope constant and $Aq(\theta)$ is the envelope function. Sampling proceeds in two stages. In the first stage, a $\tilde{\theta}$ is drawn from $q(\theta)$ independently of u that is drawn from U(0, 1). In the second stage, $\tilde{\theta}$ is either accepted or rejected according to the following rule:

- Accept: When $u \leq p(\tilde{\theta})/Aq(\tilde{\theta})$, $\tilde{\theta}$ is accepted as a value from $p(\theta)$.
- Reject: When $u > p(\tilde{\theta})/Aq(\tilde{\theta}), \tilde{\theta}$ is rejected.

The above two steps are an algorithmic way of saying – Accept $\tilde{\theta}$ with probability $p(\tilde{\theta})/Aq(\tilde{\theta})$ and reject with probability $1 - p(\tilde{\theta})/Aq(\tilde{\theta})$.

[**Why does this scheme generate samples from our target distribution** $p(\theta)$**?** The explanation lies in verifying the resulting cumulative distribution function (CDF): $P(\tilde{\theta} \leq \theta \mid \tilde{\theta} \text{ is accepted}) = \frac{P(\tilde{\theta} \leq \theta, \tilde{\theta} \text{ is accepted})}{P(\tilde{\theta} \text{ is accepted})}$ Now, $P(\tilde{\theta} \leq \theta, \tilde{\theta} \text{ is accepted}) = \int_{-\infty}^{\theta} q(\tilde{\theta}) \frac{p(\tilde{\theta})}{Aq(\tilde{\theta})} d\tilde{\theta} = \int_{-\infty}^{\theta} \frac{p(\tilde{\theta})}{A} d\tilde{\theta}$. Then $P(\tilde{\theta} \text{ is accepted}) = \int_{-\infty}^{\infty} P(\tilde{\theta} \leq \theta, \tilde{\theta} \text{ is accepted}) = \int_{-\infty}^{\infty} \frac{p(\tilde{\theta})}{A} d\tilde{\theta} = \frac{1}{A}$. Combining the two, $P(\tilde{\theta} \leq \theta \mid \tilde{\theta} \text{ is accepted}) = \int_{-\infty}^{\theta} p(\tilde{\theta}) d\tilde{\theta}$, which is the CDF under the target distribution.]

Importance sampling: It was originally proposed to obtain simulation consistent estimates. If you have sampled $\theta_1, \ldots, \theta_K$ from $q(\theta)$ instead of the original distribution $p(\theta)$, then an approximate estimate $E(t(\theta)) \approx \sum_{k=1}^{K} t(\theta_k) \frac{w_k}{\sum_k w_k}$, where $w_k = \frac{p(\theta_k)}{q(\theta_k)}$.

The specific steps are:

- First stage: Draw J (with J large) independent values $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_J\}$ from $q(\theta)$ and calculate weights $w_j = w(\theta_j)(j = 1, \ldots, J)$ as above. This defines a multinomial distribution, with categories defined by the sampled θ values and associated probabilities $\mathbf{w} = (w_1, w_2, \ldots, w_J)$.
- Second stage: Take a sample $\boldsymbol{\zeta}$ of size K from $1 : J$ i.e. draw from $Mult(K, \mathbf{w})$.

Finally, the drawn samples are $\boldsymbol{\theta}[\boldsymbol{\zeta}]$.

Now, we show that the samples from Accept/Reject and Importance sampling show the same numeric density.

```
library("AR")

lambdaden <- function(lambda){
  den      <- dnorm(log(lambda), mean = 0, sd = 1)

  out <- den
  return(out)
}

#A good choice for the proxy distribution is gamma with parameters 1,1 that will match with
#mean of the log-normal(0, 1).

samples <- AR.Sim(n=3000, lambdaden, Y.dist = "gamma", Y.dist.par = c(1, 1))
plot(density(samples))

#generate from log-normal(2, 1)
logx <- rnorm(1000, 0, 1)
xsam <- exp(logx)

#Or

ysam = rlnorm(1000, meanlog = 0, sdlog = 1)

points(density(xsam), type='l', col=2)
points(density(ysam), type='l', col=3)
```

```
######Importance sample#####
J <- 1000000
K <- 3000 #Need to be K << J, K is the number of samples you finally want
#Sample from the instrument distribution
gammasamples <- rgamma(J, 1, 1)

#Importance weights
weights      <- unlist(lapply(gammasamples, lambdaden))/dgamma(gammasamples, 1, 1)
weights <- weights/sum(weights)

#Generate K samples using Importance sampling
postsamplesIS <- gammasamples[sample(1:J, K, prob = weights)]

points(density(postsamplesIS), col=2, type = ’l’)
```

A totally fictious example showing why large J is needed for Importance sampling:

Say we want to sample from Normal$(0, .5^2)$ and use Normal$(1, 1^2)$ as the instrumental variable. Intuitively, the density of the values that are highly probable under Normal$(0, .5^2)$ will be very small under the instrument Normal$(1, 1^2)$ as most values under Normal$(1, 1^2)$ will be concentrated around 1. In this example vary J and check the p-value of `ks.test`. This is a non-parametric test, called Kolmogorov-Smirnov test which tests the null that the two vector are distributionally same. Small p-value means they are not same. When J is small, in most cases p-values will be small. As you increase J, it will get larger. Here I am applying the test on two sets of samples, one coming from IS and the other one directly applying `rnorm`.

```
for(rep in 1:10){
J <- 1000
K <- 300 #Need to be K << J, K is the number of samples you finally want
#Sample from the instrument distribution
normal11samples <- rnorm(J, 1, 1)

#Importance weights
weights      <- dnorm(normal11samples, 0, 0.5)/dnorm(normal11samples, 1, 1)
weights <- weights/sum(weights)

#Generate K samples using Importance sampling
postsamplesIS <- normal11samples[sample(1:J, K, prob = weights)]
print(ks.test(postsamplesIS,rnorm(K, 0, 0.5)))

plot(density(rnorm(K, 0, 0.5)))
points(density(postsamplesIS), col=2, type = 'l')

}

for(rep in 1:10){
J <- 5000
K <- 300 #Need to be K << J, K is the number of samples you finally want
#Sample from the instrument distribution
normal11samples <- rnorm(J, 1, 1)

#Importance weights
weights      <- dnorm(normal11samples, 0, 0.5)/dnorm(normal11samples, 1, 1)
weights <- weights/sum(weights)

#Generate K samples using Importance sampling
postsamplesIS <- normal11samples[sample(1:J, K, prob = weights)]

print(ks.test(postsamplesIS,rnorm(K, 0, 0.5)))

plot(density(rnorm(K, 0, 0.5)))
points(density(postsamplesIS), col=2, type = 'l')

}
```

2 Multivariate case

When we have multiple parameters, we use method of composition (MoC) for sampling.

Method of composition: $P(A, B, C) = P(A \mid B, C)P(B \mid C)P(C)$ or more generally, $P(A_1, A_2, \ldots, A_K) = P(A_1 \mid A_2, \ldots, A_K)P(A_2 \mid A_3, \ldots, A_K)\ldots P(A_K)$. However, this decomposition is not unique. We can follow any sequence of A_i's while writing down the decomposition. For example, $P(A, B, C) = P(A \mid B, C)P(B \mid C)P(C) = P(A \mid B, C)P(C \mid B)P(B) = P(B \mid A, C)P(A \mid C)P(C) = P(B \mid A, C)P(C \mid A)P(B) = P(C \mid A, B)P(A \mid B)P(B) = P(C \mid A, B)P(B \mid A)P(A)$.

If there are more than one parameter, you need to draw joint samples for inference. Say θ_1 and θ_2 are two parameters. Examples include normal distribution with mean and sigma both unknown, generalized linear models with at least 2 predictors. We use method of composition to write 1) $P(\theta_1, \theta_2) = P(\theta_1 \mid \theta_2)P(\theta_2)$ or 2) $P(\theta_1, \theta_2) = P(\theta_2 \mid \theta_1)P(\theta_1)$.

Which of the above two possible decomposition between 1) and 2) to be selected for sampling will depend on which of the following two integrals is easy to do.

In the above decomposition $P(\theta_2) = \int P(\theta_1, \theta_2)d\theta_1$, $P(\theta_1) = \int P(\theta_1, \theta_2)d\theta_2$.

If it is easy to integrate both of the two integrals, we can draw samples of θ_1 and θ_2 completely independently from $P(\theta_1)$ and $P(\theta_2)$ respectively.

However, that's not always possible. The reason behind this decomposition is sampling. To draw a joint sample of (θ_1, θ_2) from $P(\theta_1, \theta_2)$ we can first draw a sample of θ_1 from $P(\theta_1)$, which is called "marginal" distribution of θ_1. Then we draw our sample of θ_2 from the "conditional" distribution $P(\theta_2 \mid \theta_1)$.

Let $(X, Y) \sim \text{MVT}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where
$\boldsymbol{\mu} = \begin{pmatrix} \mu_X \\ \mu_Y \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_X^2 & \rho\sigma_X\sigma_Y \\ \rho\sigma_X\sigma_Y & \sigma_Y^2 \end{pmatrix}.$

We want to generate (X, Y) for given values of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. We consider to decompose $P(X, Y) = P(X \mid Y)P(Y)$

Marginal of Y is Normal(μ_Y, σ_Y^2) and

The conditional distribution of X given Y is
$X \mid Y = a \ \sim \ \text{Normal}\left(\mu_X + \frac{\sigma_X}{\sigma_Y}\rho(a - \mu_Y),\, (1 - \rho^2)\sigma_X^2\right).$

```
muX=170; muY=70; sX=10; sY=5; rho=0.8
X <- 0 ; Y <- 0

#Generate Y from marginal distribution
Y <- rnorm(1000, muY, sY)

#Generate X from conditional X|Y
X <- rnorm(1000,muX+(sX/sY)*rho*(Y-muY),sqrt((1-rho^2)*sX^2))
```

```
out <- cbind(x=X,y=Y)


colMeans(out)
cor(out[,1], out[,2])

apply(out, 2, sd)


acf(out[,1])
acf(out[,2])
```

Thus, to apply MoC, we need to be able to get marginal distribution for one of the parameters. It might become very hard. However, from the `acf` plots, it is clear that the samples are independent.

3 MCMC

Thus, to tackle such problems, Markov chain Monte Carlo is used. It is sampling method which is again developed by physicists like Ulam and Von Neuman [3]. 'Monte Carlo' is a fancy name for sampling. 'Markov chain' is the new part. In general, if any sequence of events have lag-1 dependence, it is called a Markov chain. The distribution of x[i+1] depended on x[i]. It is called lag-1 due to that difference of 1. It would have been lag-2 dependence if x[i+1] depended on x[i] as well as x[i-1] or x[i-1] alone. Like daily temperature usually exhibit such dependence. The temperature of Tuesday is expected to depend on that of Monday. Then it will be lag-1. However, if temperature of Tuesday is expected to depend on that of Monday and Sunday. Then it will be lag-2. There may be higher order dependence too, meaning present observation may depend on longer history.

More generally, such interdependent sequence of events is called a Markov chain. The value transit to the other based on some transition probability, which is the key part.

MCMC requires lesser mathematical derivations than MoC and thus is more popular. However, lesser math comes at a price. You need to perform a lot of post-hoc analysis to ensure 'correctness' of the samples.

In other words, MCMC is a coordinate-wise sampling method for complex multivariate distributions, similar to coordinate descent for optimization.

Autocorrelation: Correlation computes the association between two different variables like age and height or height and weight. Autocorrelation computes the association within itself. It is in general applied to the datasets that are collected sequentially, like daily temperature data, stock market data, etc. In temperature data, for example, we may be interested to know 'What the association between the temperatures on the two consecutive days?' Say we have daily temperature data as 50,53,52,49,48,49,53,54,51. How do you answer the above Qn?

Autocorrelations are defined in terms of lag-'number'. Lag-1 autocorrelation is the correlation $\gamma(1)=\text{cov}(X_t X_{t-1})$. Similarly, Lag-$h$ autocorrelation is the correlation $\gamma(h)=\text{cov}(X_t X_{t-h})$. In R, `acf()` computes these correlations and also provide results on their significance (will see).

In an ideal sample, all the samples are expected to be independent. Like for all other sampling methods, we previously discussed. This means that $\gamma(h)=0$ or statistically insignificant for any $h>0$.

However, by construction, MCMC samples are not fully independent samples of the give multivariate distribution. There is some dependence. We see that below example.

3.1 Gibbs sampling:

Gibbs sampling requires some mathematical derivations, but much less than MoC.

We know how to generate from univariate normal. We want to use that to generate from bivariate normal.

The conditional distribution of X given Y is

$$X \mid Y=a \;\sim\; \text{Normal}\left(\mu_X+\frac{\sigma_X}{\sigma_Y}\rho(a-\mu_Y),\,(1-\rho^2)\sigma_X^2\right).$$

and conditional Y given X is $Y \mid X=b \;\sim\; \text{Normal}\left(\mu_Y+\frac{\sigma_Y}{\sigma_X}\rho(b-\mu_X),\,(1-\rho^2)\sigma_Y^2\right).$

```
muX=170; muY=70; sX=10; sY=5; rho=0.8
X <- 0 ; Y <- 0
X[1] <- rnorm(1,muX,sX) #init value for x_0
```

```
Total_itr <- 2000

for(i in 1:Total_itr){
  # sampling from Y|X
  Y[i]   <- rnorm(1,muY+(sY/sX)*rho*(X[i]-muX),sqrt((1-rho^2)*sY^2))

  # sampling from X|Y
  X[i+1] <- rnorm(1,muX+(sX/sY)*rho*(Y[i]-muY),sqrt((1-rho^2)*sX^2))
}
out <- cbind(x=X[-1],y=Y)

#The samples after discarding a few initial samples
outp <- out[-c(1:500), ]

acf(outp[,1])
acf(outp[,2])

colMeans(outp)
cor(outp[,1], outp[,2])

apply(outp, 2, sd)

#The samples after discarding a few initial samples and thinning
outp <- out[500+5*(1:300),]

acf(outp[,1])
acf(outp[,2])

cor(outp[,1], outp[,2])
```

In the above example, we are moving in the following manner:
$(X^0, Y^0) \to (X^1 \to Y^1) \to (X^2 \to Y^2) \cdots$

Finally, $\{X^{501}, \ldots, X^{1000}\}$ are our samples of X after confirming that these are 'good' samples. Similarly, for Y.

What we see different above?:

- The samples are generated by conditioning on the rest of the parameters.

- Thus, the samples become dependent on the previously generated samples. (Clue: we are using a while loop for sampling, which can not parallelized.)

Potential pitfall:

- Will my estimate depend on the starting value?

- For statistical inference, we need the samples to be independent. However, these samples are dependent.

- The sampler can not be parallelized (like for MoC, we could use `apply` while sampling Y for each X and also we generated all the samples of X together as used the marginal distribution for X, not conditional).

Elaboration of first point: The estimate may depend on initialization if the likelihood is multi-modal. It might get trapped in a local mode. Thus, it is advisable to run the MCMC chains for multiple starting points. This is also fairly common in any optimization routine.

Elaboration of second point: We start with some Y^0 and X^0. Given Y^0, we sample/update X and move from $X^0 \to X^1$. This X^1 is the sample . Next, given this new X^1, we draw a sample of Y and move from $Y^0 \to Y^1$. This Y^1 is the sample

In the next iteration: We generate X given Y^1 and further move from $X^1 \to X^2$.

If we generalize above step for K-many parameters, we simply update each of the K-parameters one by one given all the other. For example, our $\mathbf{x} = (x_1, \ldots, x_{10})$. We can then sample x_1 from $P(x_1 \mid x_2, \ldots, x_{10}, Y)$, x_2 for $P(x_2 \mid x_1, x_3, \ldots, x_{10}, Y)$, $\ldots$.

Elaboration of third point: Why we need to do it sequentially but not parallelly like in the past? The answer to that is in the probability. We have $P(X, Y) = P(X \mid Y)P(Y)$. But we are generating sample from conditional distributions for both of the two parameters i.e. we are sampling X from $P(X \mid Y)$ and sampling Y from $P(Y \mid X)$, AND $P(X, Y) \neq P(X \mid Y)P(Y \mid X)$.

This has some commonality with standard multiparameter optimization routines. But they are conceptually not the same. In any optimization routine, we update the parameters sequentially in a loop. MCMC can be considered as a sampling-based alternative of the optimization. However, MCMC requires much larger number of iterations to converge.

3.2 General Metropolis-Hastings (MH)

It may not be possible to arrive at a known distribution class for all the full conditional distributions. For such situations, we consider the MH algorithm for sampling.

How does it work? Let't take the previous example. Say we are at *t-th* iteration and $Y = Y^t$. We want to generate $(t+1)$-*th* sample of X.

- Let X^t be the value of X at *t-th* MCMC iteration.
- For $t+1$-*th* iteration, we propose a new value of X_c (which is called candidate).
- Next would be to decide whether this new value should be considered as a new sample/update for X. If it gets selected, then we set $X^{t+1} = X_c$ otherwise set $X^{t+1} = X^t$.

How do we propose this new value and decide whether to accept it or not?

- Transition: Choose a probability distribution with one of the parameters being X^t and the expectation under this distribution should ideally be X^t. Example: If we choose Normal(X^t, ϵ^2). The expectation of this distribution is X^t. This distribution is called *Transition probability* and denoted by q. The probability of transitioning from X^t to X_c is denoted by $q(X_c \mid X^t)$ when $q(\cdot \mid X^t) = \text{Normal}(X^t, \epsilon^2)$, we can calculate $q(X_c \mid X^t)$=`dnorm`($X_c \mid X^t$, 0, ϵ)
- Acceptance: We are not accepting all the updates at each move. It is a probabilistic move. It relies on the acceptance probability, which is computed as $p = \frac{P(X_c,Y^t)}{P(X^t,Y^t)} \frac{q(X^t|X_c)}{q(X_c|X^t)}$, where X_c is the candidate-update of X.

The above acceptance rate is derived following the principle of detailed balance, which is a concept for studying kinetic systems in Physics. Here we are running an iterative scheme to generate samples. The expectation is that these samples will be coming from the 'same' distribution after some time. This distribution is called the stationary distribution or the equilibrium distribution. Now to achieve such distribution, we need this iterative scheme to be in equilibrium, which requires the process to be reversible. The reversibility of the Markov chain is ensured by the detailed balance equation, which leads to the above acceptance rate. [1] is great paper providing an easy-to-read theoretical explanation on this topic.

Gibbs sampling is a special case of General MH sampling where we were accepting all the proposals as **we can show acceptance probability of a Gibbs sampler is 1**.

3.3 Random walk MH:

MH sampling requires no mathematical derivations. You just need to be able to write down the likelihood. MH sampling can be applied for ANY probability distribution, no matter how complex that might look. Well, it comes at a price. We study that later.

Random walk MH sets the transition probability to Normal distribution. Hence, for random-walk MH, the q is always Normal. Then $q(X^t \mid X_c) = q(X_c \mid X^t)$ which simplifies our acceptance probability.

The optimal acceptance rate for MH sampler is 30% for Gaussian-type models for univariate random variables. This result is not generalizable. Thus, it is recommended to maintain an acceptance with a pre-specified range like 10%-40% or 20%-50%. Different papers may suggest different ranges based on their simulation studies.

```
muX=170; muY=70; sX=10; sY=5; rho=0.8
X <- 0 ; Y <- 0
X[1] <- rnorm(1,muX,sX) #init value for x_0

Total_itr <- 10000

#The mean and standard deviation of our multivariate normal distribution
muXY <- c(170, 70)
sigmaXY <- cbind(c(sX^2, rho*sX*sY), c(rho*sX*sY, sY^2))

log_det_sigmaXY <- as.numeric(determinant(sigmaXY)$modulus)
sigmaXYinv <- solve(sigmaXY)

#The standard deviation of the MH proposal
sd1 <- 0.1
sd2 <- 0.1

#To count the number of MH candidates getting accepted.
ar1 <- 0
ar2 <- 0

for(i in 2:Total_itr){
  #M-H step candidate for Y
  yc <- Y[i-1] + rnorm(1, sd=sd1)
```

```
#new log-density
newvec <- c(X[i-1], yc)
llnew <- -log(2*pi)/2-log_det_sigmaXY/2-t(newvec-muXY)%*%sigmaXYinv%*%(newvec-muXY)

#old log-density
oldvec <- c(X[i-1], Y[i-1])
llold <- -log(2*pi)/2-log_det_sigmaXY/2-t(oldvec-muXY)%*%sigmaXYinv%*%(newvec-muXY)

R <- llnew - llold

if(is.na(R)) {R = -Inf}
if(is.nan(R)) {R = -Inf}
logu <- log(runif(1))
if (logu < R)
{
  Y[i] <- yc #new sample of Y is yc
  ar1 <- ar1 + 1
}
if (logu >= R) {
  Y[i] <- Y[i-1]}


#M-H step candidate for Y
xc <- X[i-1] + rnorm(1, sd=sd2)

#new log-density
newvec <- c(xc, Y[i]) #We use the new value of Y
llnew <- -log(2*pi)/2-log_det_sigmaXY/2-t(newvec-muXY)%*%sigmaXYinv%*%(newvec-muXY)/2

#old log-density
oldvec <- c(X[i-1], Y[i])
llold <- -log(2*pi)/2-log_det_sigmaXY/2-t(oldvec-muXY)%*%sigmaXYinv%*%(newvec-muXY)/2

R <- llnew - llold

if(is.na(R)) {R = -Inf}
if(is.nan(R)) {R = -Inf}
logu <- log(runif(1))
```

```
  if (logu < R)
  {
    X[i] <- xc #new sample of X is xc
    ar2 <- ar2 + 1
  }
  if (logu >= R) {
    X[i] <- X[i-1]}

  #After each 100 iterations
  if(i %% 100 == 0){
    #Adjust the standard deviation to maintain an acceptance rate within (0.2,0.5)
    ar <- ar1 / i
    cat(ar, "for Y")
    if(ar>.50){sd1 <- sd1 * 2}
    if(ar<.20){sd1 <- sd1 / 2}

    #Adjust the standard deviation to maintain an acceptance rate within (0.2,0.5)
    ar <- ar2 / i
    cat(ar, "for X")
    if(ar>.50){sd2 <- sd2 * 2}
    if(ar<.20){sd2 <- sd2 / 2}
  }
}
out <- cbind(x=X,y=Y)

burn <- 5000

#The samples after discarding a few initial samples
outp <- out[(burn+1):Total_itr, ]

Xp <- outp[, 1]
Yp <- outp[, 2]

colMeans(outp)
cor(outp[,1], outp[,2])

apply(outp, 2, sd)
```

```
#Convergence diagnostic
acf(X[(burn+1):Total_itr], lag.max = 100)
acf(Y[(burn+1):Total_itr], lag.max = 100)

M <- 60
Total_samples <- Total_itr-burn

#Samples to be taken after thinning
ind <- M*(1:(Total_samples/M))

#Thinned samples have 0 autocorrelation, thus these samples are independent
acf(Xp[ind])
acf(Yp[ind])
```

Convergence diagnostics
Things to check:

1. Autocorrelation of the chain.
2. Thin the generated samples to reduce the autocorrelation.
3. Whether the performance of the sampler depend on the starting value. (It is often good to devise a strategy for initialization of the parameters and make that a default.)

Points to note

- Random walk MH is a special type of MH. This the most used one as for this case, we have $q(X_c \mid X^t) = P(X^t \mid X_c)$.
- However, random walk updating requires the parameter to be unrestricted which is why we might need to transform the variable to an unrestricted. But then the joint density needs to be adjusted using the Jacobian. Say we want to sample (θ_1, θ_2) from $f(\theta_1, \theta_2)$, where θ_2 is continuous, but bounded. Thus, we consider a new variable λ_2 such that $\theta_2 = v(\lambda_2)$ and λ_2 is unbounded. Let λ_{2c} be a candidate of λ_2. Then the acceptance probability will become $= \frac{g(\theta_1, \lambda_{2c})}{g(\theta_1, \lambda_2)}$, where $g(\theta_1, \lambda_2) = |v'(\lambda_2)| f(\theta_1, v(\lambda_2))$, the joint density of (θ_1, λ_2).
- The optimal acceptance rate is around 0.35-0.45 for random walk MH and it decreases with the dimension of the parameter. Usually keeping it between 0.3-0.5 produces good results.

- In random-walk MH, the acceptance is solely governed by the difference in log-likelihood and thus making it increasingly similar to optimization routines.

Table 1: Transformation for making a restricted random variable unrestricted

Given variable	Transformation	Inverse transformation	derivative
If $\theta > 0$	Take log transformation and set $\theta_1 = \log(\theta)$	$\theta = \exp(\theta_1)$	$\exp(\theta_1)$
Bounded: If $\theta \in (a, b)$	Set $\lambda_1 = \log(\theta'/(1-\theta'))$ where $\theta' = \frac{\theta-a}{b-a}$	$\theta = (b-a)\frac{\exp(\lambda_1)}{1+\exp(\lambda_1)} + a$	$(b-a)\frac{\exp(\lambda_1)}{(1+\exp(\lambda_1))^2}$ (Verify!)

4 Some variations of the above MH sampling

4.1 Block update

We can update (X, Y) together.

```
muX=170; muY=70; sX=10; sY=5; rho=0.8
X <- 0 ; Y <- 0
X[1] <- rnorm(1,muX,sX) #init value for x_0

Total_itr <- 10000

#The mean and standard deviation of our multivariate normal distribution
muXY <- c(170, 70)
sigmaXY <- cbind(c(sX^2, rho*sX*sY), c(rho*sX*sY, sY^2))

log_det_sigmaXY <- as.numeric(determinant(sigmaXY)$modulus)
sigmaXYinv <- solve(sigmaXY)

#The standard deviation of the MH proposal
sd <- 0.1

#To count the number of MH candidates getting accepted.
ar <- 0

for(i in 2:Total_itr){
  #M-H step candidate for Y and X together
  yc <- Y[i-1] + rnorm(1, sd=sd)
  xc <- X[i-1] + rnorm(1, sd=sd)
  #new log-density
  newvec <- c(xc, yc)
  llnew <- -log(2*pi)/2-log_det_sigmaXY/2-t(newvec-muXY)%*%sigmaXYinv%*%(newvec-muXY)

  #old log-density
  oldvec <- c(X[i-1], Y[i-1])
  llold <- -log(2*pi)/2-log_det_sigmaXY/2-t(oldvec-muXY)%*%sigmaXYinv%*%(newvec-muXY)

  R <- llnew - llold

  if(is.na(R)) {R = -Inf}
  if(is.nan(R)) {R = -Inf}
  logu <- log(runif(1))
  if (logu < R)
  {
    Y[i] <- yc #new sample of Y is yc
    X[i] <- xc
    ar <- ar + 1
  }
  if (logu >= R) {
    Y[i] <- Y[i-1]; X[i] <- X[i-1]}

  #After each 100 iterations
  if(i %% 100 == 0){
    #Adjust the standard deviation to maintain an acceptance rate within (0.2,0.5)
    ar <- ar / i
    cat(ar, "for Y and X")
    if(ar>.50){sd <- sd * 2}
    if(ar<.20){sd <- sd / 2}
  }
}
out <- cbind(x=X,y=Y)

burn <- 5000

#The samples after discarding a few initial samples
outp <- out[(burn+1):Total_itr, ]
```

```
Xp <- outp[, 1]
Yp <- outp[, 2]

colMeans(outp)
cor(outp[,1], outp[,2])

apply(outp, 2, sd)


#Convergence diagnostic
acf(X[(burn+1):Total_itr], lag.max = 100)
acf(Y[(burn+1):Total_itr], lag.max = 100)

M <- 60
Total_samples <- Total_itr-burn

#Samples to be taken after thinning
ind <- M*(1:(Total_samples/M))

#Thinned samples have 0 autocorrelation, thus these samples are independent
acf(Xp[ind])
acf(Yp[ind])
```

4.2 Adaptive block MH

In the case of above block MH, the candidate is generated as `c(yc, xc) <- c(Y[i-1], X[i-1]) + rnorm(2, sd=sd)` which may be replaced by `c(yc, xc) <- c(Y[i-1], X[i-1]) + e`, where $\mathbf{e} \sim \text{MVN}(0, \sigma^2 \mathbf{R}^{(i)})$ and $\mathbf{R}^{(i)}$ is a correlation matrix which can be calculated from the generated past $i-1$ samples [2].

5 Data augmentation

In case of missing observations and mixed effect models, data augmentation is used while conducting Bayesian inference. If there are missing observations, then within MCMC as Gibbs sampling step we can impute the missing observations by sampling those from the original model. Let $P(\mathbf{y}_{\text{obs}}, \mathbf{y}_{\text{miss}}|\boldsymbol{\theta})$ be the joint likelihood of the data. Then we can sample $P(\mathbf{y}_{\text{miss}} \mid \mathbf{y}_{\text{obs}}, \boldsymbol{\theta})$ and subsequently $P(\boldsymbol{\theta} \mid \mathbf{y}_{\text{miss}}, \mathbf{y}_{\text{obs}})$. Then repeat. The mixed-effect linear models within a Bayesian setting are fitted in the same way.

References

[1] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335, 1995.

[2] Heikki Haario, Eero Saksman, and Johanna Tamminen. An adaptive metropolis algorithm. *Bernoulli*, pages 223–242, 2001.

[3] Christian Robert and George Casella. A short history of markov chain monte carlo: Subjective recollections from incomplete data. *Statistical Science*, 26(1):102–115, 2011.