How to Integrate using Statistics?

An Introduction to Monte Carlo Methods and MCMC

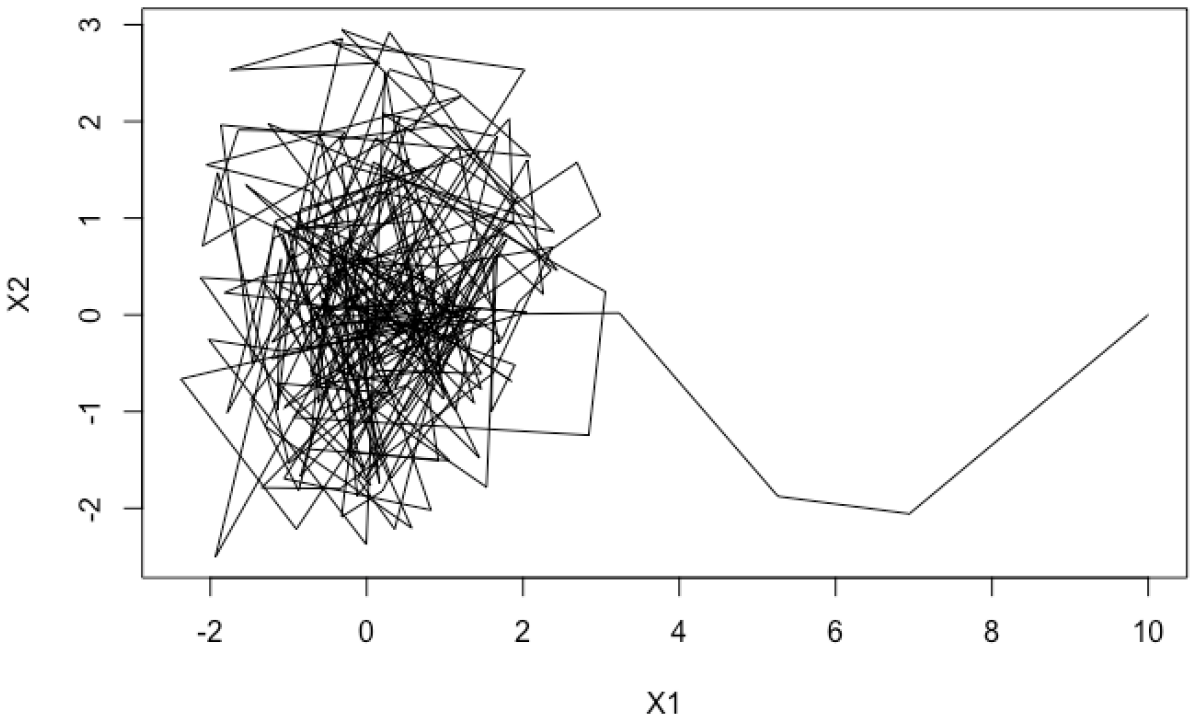
Zhang Ruiyang

Contents

Preface

This set of notes is prepared as supplementary material for the talk I gave at UCL Undergraduate Math Colloquium.

Abstract: Integration is no stranger to us, and we have learned a lot of tricks and theories about them. However, in a lot of the real life applications, we would come across integrals that are hard to compute and there might not even be a good expression of the antiderivative of the integrand. Such integrals, known as the intractable integrals, could in fact be calculated (or approximated) via statistical methodologies known as the Monte Carlo methods. In this talk, we will be studying some of the Monte Carlo methods as well as some Markov Chain Monte Carlo (MCMC) algorithms such as MH, MALA, and HMC. Basic knowledge of Markov chains, Law of Large Numbers and Central Limit Theorem would be helpful, but the talk will ultimately be self-contained.



1 Monte Carlo: Why and How

Monte Carlo methods arises from the strong urge to compute integrals in Statistics (and other disciplines too). There are a lot of integrals to be computed, yet at the same time they are hard to do.

For the first part of the statement, I will showcase an examples of important integrals to illustrate my point. For a (continuous) random variable X with cumulative distribution function (cdf) F and probability density function (pdf) f, if we want to compute its expectation, we would have

$$\mathbb{E}(X) = \int x dF = \int x f(x)\ dx.$$

To generalise it slightly, for a function of X, say $g(X)$, its expectation will be

$$\mathbb{E}(g(X)) = \int g(x) dF = \int g(x) f(x)\ dx.$$

Depending on how complicated g and f are, the above integrals can be either very trivial or very troublesome to deal with.

There are other types of integrals that we will encounter in Statistics, say for hypothesis testing when we are computing the p-value of a particular set of samples or for Bayesian inference when we are calculating the posterior distribution and some expectations afterwards. The example above should be enough to make my case, that integrals are important.

Not all integrals are created equal, and some are harder than others. In Mathematics, and probably your Calculus classes, one would want to have a closed form expression for the antiderivatives of any integral you are working with, and obtain every single results using only pen and paper. Integrals one encounter in real life might get more tricky. A classical example would be the following:

$$\int_a^b \cos(x) e^{-x^2/2}\ dx.$$

There is no elementary function being the antiderivative of the integrand above, so the hope of finding its antiderivative then find the difference of the antiderivative at the two end points (using the Fundamental Theorem of Calculus) is futile. For such integrals that do not exist an antiderivative in the form of an elementary function, we will call them **intractable**, and find other ways to work with them.

Of course, if you are familiar with probability theory, especially the normal distribution, you may notice that the above integrand is $\cos(x)$ times the pdf of a standard normal distribution. If we have the integral as $\int \cos(x) e^{-x^2/2}\ dx$, it will become the expectation of $\cos(X)$ where $X \sim N(0, 1)$. However, this is not enough to give us an answer to the question. Well, not yet!

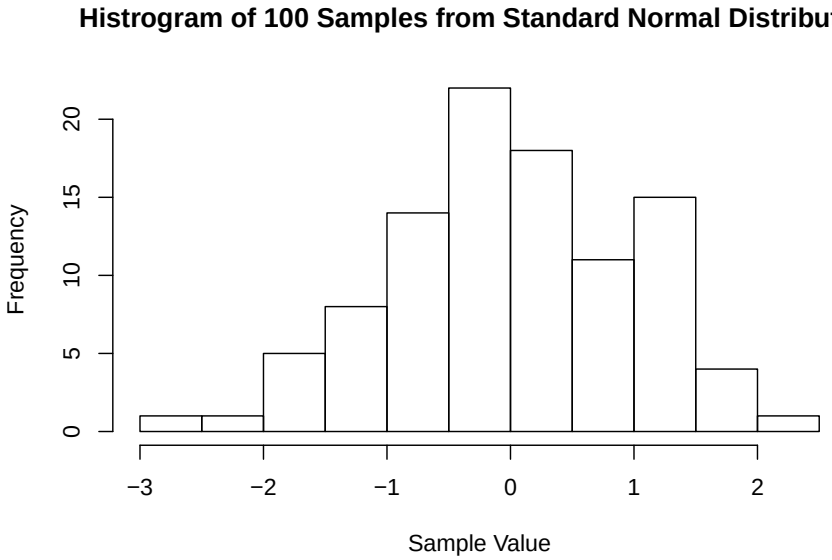
Here is one proposal (the **Monte Carlo method**). Since we know that the integrand is the expectation of a function of a random variable following a known distribution, if we can obtain some (independent) samples from the distribution and find its mean after applying the function to all the samples, then we would have computed the desired integral after multiplying the mean by $\sqrt{2\pi}$. For $\int_a^b \cos(x) e^{-x^2/2}\ dx$, we will view it as $\int I_{[a,b]}(x) \cos(x) e^{-x^2/2}\ dx$ where $I_{[a,b]}(x)$ is the indicator function that takes 1 when $x \in [a, b]$ and 0 otherwise. So, the integral is computed by taking all the samples between a and b and summing them after cos, dividing them by the total number of samples obtained, then multiplying the mean by $\sqrt{2\pi}$. A demonstration of this strategy will be shown below.

First, we need to get some samples from the standard normal distribution. This is easy to achieve using R (although not easy in general). The following is a list of 100 samples (2 dp) of standard normal.

```
##   [1]  1.16 -0.59  1.79 -1.33 -0.45  0.57 -2.89 -0.87 -0.46 -0.56 -0.02 -0.15
##  [13] -0.63  1.32 -1.52 -0.44  0.97  0.03 -0.09  0.39  0.24 -0.14  0.72  0.37
##  [25] -0.24 -1.47 -0.60 -1.15 -2.47 -0.61 -0.22  1.59  1.56  1.11 -1.10 -1.86
```

```
##  [37] -0.91  1.25  0.09  0.42 -0.82 -1.54  0.56 -0.37 -1.05  0.02  0.88  0.88
##  [49]  1.03 -0.38  1.10 -0.03  0.19  1.34  0.73  0.06  1.33 -0.41 -0.82  0.36
##  [61]  0.06 -0.07 -0.38 -0.87  1.20 -1.66  1.17 -1.07  0.91 -1.32  0.66 -0.66
##  [73]  0.99 -0.68  1.12  0.39 -0.15 -0.47 -1.36 -0.03 -0.24  1.92  0.10  0.09
##  [85]  0.19 -0.64 -0.04  0.11  1.21  1.44  2.21 -1.97  0.56 -0.22 -0.07  0.27
##  [97]  1.29  1.10 -0.60  0.16
```

To get a rough sense that this is indeed from standard normal, we plot a histogram of these numbers, and we can observe a normal-ish trend in it.



Next, we would apply cos to them, and the samples will become the following.

```
##   [1]  0.40  0.83 -0.21  0.24  0.90  0.84 -0.97  0.65  0.90  0.85  1.00  0.99
##  [13]  0.81  0.25  0.05  0.91  0.56  1.00  1.00  0.93  0.97  0.99  0.75  0.93
##  [25]  0.97  0.10  0.83  0.41 -0.79  0.82  0.98 -0.02  0.01  0.45  0.46 -0.29
##  [37]  0.61  0.32  1.00  0.91  0.68  0.03  0.85  0.93  0.50  1.00  0.64  0.64
##  [49]  0.52  0.93  0.45  1.00  0.98  0.23  0.74  1.00  0.24  0.92  0.68  0.94
##  [61]  1.00  1.00  0.93  0.64  0.37 -0.09  0.39  0.48  0.61  0.25  0.79  0.79
##  [73]  0.55  0.78  0.43  0.93  0.99  0.89  0.21  1.00  0.97 -0.34  1.00  1.00
##  [85]  0.98  0.80  1.00  0.99  0.35  0.13 -0.60 -0.39  0.85  0.98  1.00  0.96
##  [97]  0.28  0.45  0.83  0.99
```

By taking the mean of the above samples and then multiplying the mean by $\sqrt{2\pi}$, we get $\int \cos(x)e^{-x^2/2}\,dx \approx 1.44$. You can check this results using your favourite integral calculator and see that 1.44 is not too far off from it. The accuracy of the result will certainly increase if we take more samples. If we sample 10000 standard normal samples and apply the same procedure, we would get 1.53, which should be even closer to the numerical results you get from your favourite integral calculator.

Regarding the simulation carried out just now, one should note that it is applied after setting the seed to a fixed value, so that we will obtain the same results every time we run the code. By removing the set.seed() function, we will get different results each time we run it. This is only included for the consistency of the results of the notes.

At this stage, one might be curious about how valid the procedure is. Well, it is quite valid, and it has theoretical justification too. The procedure is justified using a combination of Central Limit Theorem and

Law of Large Number, and we will omit the details of the justification here. We can just accept to be true that the estimator produced by the procedure is unbiased and consistent. Interested readers could find an elaborate proof online.

Let us now recap what has happened so far. We tried to compute an integral, and we notice a portion of the integrand is very similar to the pdf of a known distribution. We then view the integral as the expectation of a function of a random variable that follows that distribution to calculate the integral. There are two points in the process that require a second thought:

(1) Will this method be valid for a more general setting with a more general integral? What if the integrand does not contain the pdf of some known distribution?

(2) Can we always obtain samples from any distributions?

The first question (or the first set of questions) can be answered fairly easily. For a general integral $\int f(x)\,dx$ where $f(x)$ is not related to some probability distribution, we can introduce a pdf $q(x)$ of some distribution and rewrite the integral as $\int [f(x)/q(x)] \cdot q(x)\,dx$. Now, this is of a more familiar form, and the method we had is applicable now. Another follow-up question would arise at this point - what $q(x)$ should we use then? Well ... this require some careful considerations, and we will not discuss it here - though it can be answered.

The second question is the one that we are more interested in here. Can we always obtain samples from any distributions? The answer is not really. In the rest of this notes, we will be studying some of the ways we can obtain (good quality) samples from certain distributions, and this question is a very active area of research in Statistics.

2 Metropolis-Hastings Algorithm

2.1 Markov Chain Basics

Markov Chain is one example of a stochastic process, which is a sequence of random variables indexed by time. The index would be $\mathbb{N}$ when we are working in discrete time, and $\mathbb{R}^{>0}$ when we are working in continuous time. We will illustrate the discrete time case here since it is easier to grasp the general idea of Markov chain and its long-run behaviour, although the continuous time version is significant in the theoretical aspect of MCMC. This section is adapted mostly from Norris' Markov chain (Norris 1998).

In this section, we will be going over the basics of Markov chain by answering three questions: (1) What is a Markov chain? (2) How does a Markov chain develop over time? (3) How will a Markov chain end?

2.1.1 What is a Markov chain?

With each Markov chain, we will be working on a **state space** I and the random variables will all give values of some element of I. Each $i \in I$ is called a state, and I will be a countable set. A **distribution** on I is a row vector $\lambda = (\lambda_i)_{i \in I}$ with $\sum \lambda_i = 1$. For each λ_i, it is defined to be the probability for the random variable to be on state i.

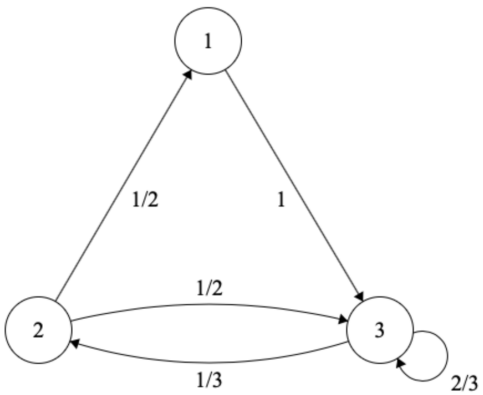
To capture the probability of moving from one state to another, we will let such a probability be p_{ij} for probability of moving from state i to state j. Notice that we are assuming this probability to stay constant regardless of time, and this property is called the **time-homogeneity**. To write in more formal language, for a Markov chain $(X_n)_{n \in \mathbb{N}}$, we have $\mathbb{P}(X_{n+1} = j | X_n = i) = p_{ij}$. All the probabilities p_{ij}, known as the one-step transition probability, are summarised into a matrix P called the transition matrix, and it has the property that each row sums to 1 since each row is a distribution.

To uniquely define chain, we would also need to know the starting point of the chain, and the starting point can be a distribution although it could be a point too (1 at that point and 0 elsewhere). That distribution is called the **initial distribution** λ, and we have had enough information about the initiation of a Markov chain, and how it moves.

A very important property of a Markov chain (and probably why it is so powerful and applicable) is the Markov property. It is the property of memoryless, meaning that the information about the past is independent of future events. A rather famous quote due to this property is "Life is like a Markov chain. Your future only depends on what you are doing now, and independent of your past." To formally write out this property, we have

$$\mathbb{P}(X_{n+1} = j | X_n = i, X_{n-1} = x_{n-1}, \cdots, X_0 = x_0) = \mathbb{P}(X_{n+1} = j | X_n = i) = p_{ij}.$$

The following is an example of a Markov chain, in graphical form.



Although we have not even mentioned anything about the construction of Markov chain diagrams, one should be able to understand rather easily what the above figure is trying to say. The state space is $\{1, 2, 3\}$, and the transition matrix is $\begin{bmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 1/2 \\ 0 & 1/3 & 2/3 \end{bmatrix}$.

2.1.2 How does a Markov chain develop over time?

We have already gotten the transition matrix P, which is the matrix with all the one-step transition probabilities. Now, how about two-steps transition, three-steps transition, ... , and n-steps transition?

If we want to go from state i to state j in two steps, we need to go from i to some intermediate k then to j. The intermediate k can be any reasonable states in the state space, since we are only concerned about the starting and ending point, as well as the number of steps. This gives us the **Chapman-Kolmogorov equation**

$$p_{ij}^{(2)} = \sum_{k \in I} p_{ik} p_{kj}$$

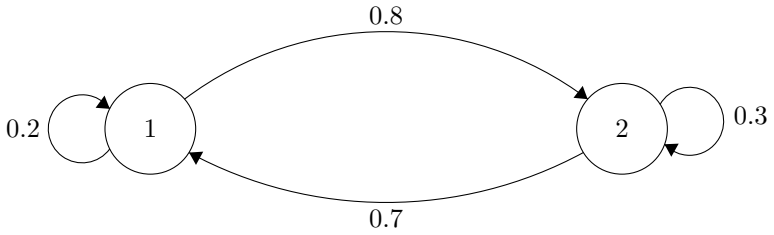
where the superscript (2) indicates the number of steps. This should not be a surprise. Next, if we recall some matrix multiplication rules from Linear Algebra, we may notice that the above equation is simply matrix multiplication of the ith row of P and jth column of P. If we generalise this for all possible combination of i and j, we would get P^2, the square of the transition matrix P which is the 2-step transition matrix.

This can be generalised to n-step, and the n-step transition matrix would simply be P^n. Using this, we can have a good idea where the chain will be at any time, give the starting point / distribution.

2.1.3 How will a Markov chain end?

How would a chain behave in the long run? This is the same as asking how would P^n be as $n \to \infty$?

We will use an example to understand the long run behaviour.



Here, the transition matrix is $P = \begin{bmatrix} 0.2 & 0.8 \\ 0.7 & 0.3 \end{bmatrix}$. Its two eigenvalues are 1 and -0.5, so we have $P = Q \begin{bmatrix} 1 & 0 \\ 0 & -0.5 \end{bmatrix} Q^{-1}$, and $P^n = Q \begin{bmatrix} 1^n & 0 \\ 0 & (-0.5)^n \end{bmatrix} Q^{-1}$ of some eigenmatrix Q.

Thus, $\lim_{n \to \infty} P^n = Q \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} Q^{-1} = \begin{bmatrix} 7/15 & 8/15 \\ 7/15 & 8/15 \end{bmatrix}$, which indicates the equilibrium distribution of the chain.

As one can see, in the long run, the chain converges to the distribution $(7/15, 8/15)$. This is quite an easy example as we only have two states taking 1 and 2. What if we have a chain that takes the entire real line, and what could the long run distribution of that chain be? You'll see.

You'll see.

2.2 MCMC General Idea

Our goal is to obtain samples following some target distribution $\pi(\cdot)$. The idea behind MCMC algorithms is that a Markov chain may converge to a distribution as the chain carries on for a while. That long-run distribution is known as the equilibrium distribution. If we can find a way to construct some sort of Markov chains with long-run distribution $\pi(\cdot)$, the values of the chain (after it converges) will then be our ideal samples. Different MCMC algorithms come up with different ways to generate the Markov chain that has long run distribution $\pi(\cdot)$.

There are a few popular MCMC algorithms with reasonable amount of theoretical justifications and empirical evidence that are frequently used in practice in varied contexts. Three of the few are Metropolis-Hastings algorithm (MH), Metropolis adjusted Langevin algorithm (MALA), and Hamiltonian Monte Carlo algorithm (HMC). We will cover MH in this chapter, and HMC in the next chapter. MALA will not be explained in this notes.

One thing to note is that MCMC has strong connections with Physics. The earliest work in this field are done by physicists, and the aforementioned Langevin and Hamiltonian are all concepts in Physics, from Statistical Mechanics and Dynamics respectively.

2.3 Metropolis-Hastings and Random Walk Metropolis

Metropolis-Hastings algorithm, as the name may have already suggested, is proposed by Metropolis and Hastings. Metropolis et al. proposed this algorithm when he and his peers in Los Alamos (Metropolis et al. 1953), and Hastings generalised the work several years later (Hastings 1970). Here, I will explain the algorithm, and illustrate one special case of it - the random walk metropolis (RWM).

The following is the algorithm:

Input: Starting value $X_0 = x$, target distribution $\pi(\cdot)$, proposal kernel $Q(\cdot, \cdot)$, number of iterations n, acceptance function $\alpha(\cdot, \cdot)$

Main: For $i \in \{0, 1, \cdots, n\}$:

1. Draw $Y \sim Q(X_i, \cdot)$
2. Draw $U \sim U[0, 1]$,
3. If $U \leq \alpha(X_i, Y)$, let $X_{i+1} = Y$. Else, let $X_{i+1} = X_i$.

Ouput: $\{X_n\}$

The acceptance function α is normally defined as

$$\alpha(x, y) = \min\left\{1, \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}\right\}$$

where $Q(x, A) = \int_A q(x, y)\ dy$ for the proposal kernel Q and π be the target distribution. This acceptance rate has been proved to be more optimal than some other choices under some setting (Peskun 1973).

As one may have already noticed, this is a very general algorithm. I will show a more specific case, the random walk Metropolis, below.

Input: Starting value $X_0 = x$, target distribution $\pi(\cdot)$, number of iterations n, acceptance rate $\alpha(\cdot, \cdot)$, tuning factor σ

Main: For $i \in \{0, 1, \cdots, n\}$:

1. Draw $Y \sim N(X_i, \sigma^2 I)$

2. Draw $U \sim U[0,1]$,

3. If $U \leq \alpha(X_i, Y)$, let $X_{i+1} = Y$. Else, let $X_{i+1} = X_i$.

Ouput: $\{X_n\}$

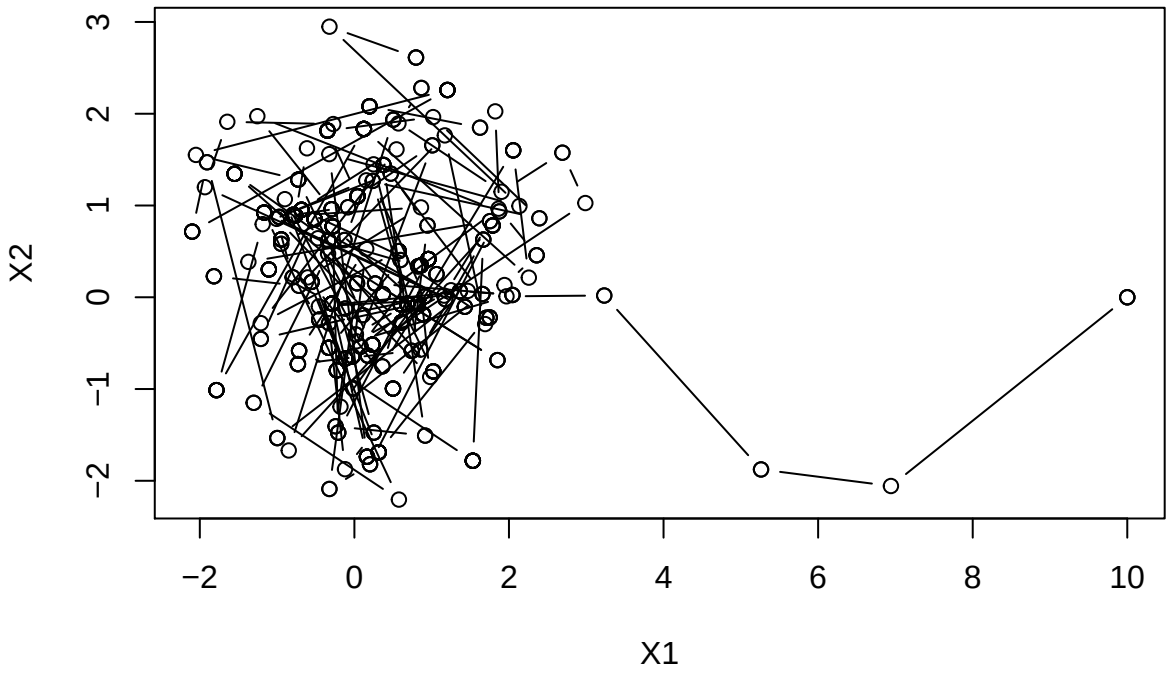
Here, the acceptance function α is normally defined as

$$\alpha(x,y) = \min\left\{1, \frac{\pi(y)}{\pi(x)}\right\}$$

since q is symmetric and $q(x,y) = q(y,x)$ for the Q in this algorithm.

Next, I will show an implementation of the RWM algorithm to sample from 2D standardised Gaussian. The starting point is (10,0), and one can see from the graph that it soon starts to converge after a few steps. The readers are encouraged to modify the code to sample from different distributions.

```
# multi-dimensional RWM code

set.seed(22)
logpi_gauss <- function(x) { dnorm(x[1],log=T)+dnorm(x[2],log=T) }
nits <- 500

# create a 1d random walk metropolis sampling function
RWM <- function(logpi, nits, h, x_curr) {
  logpi_curr <- logpi(x_curr)
  accepted <- 0
  d <- length(x_curr)
  x_store <- matrix(ncol=d, nrow=nits)


  for (i in 1:nits) {
    # propose a candidate move
    x_prop <- x_curr + h*rnorm(d)
    logpi_prop <- logpi(x_prop)

    # accept-reject
    loga <- logpi_prop - logpi_curr
    u <- runif(1)
    if (log(u) < loga) {
      x_curr <- x_prop
      logpi_curr <- logpi_prop
      accepted <- accepted + 1
    }
    x_store[i,] <- x_curr
  }

  return(list(x_store = x_store, a_rate = accepted/nits))
}

# run the function to generate a Markov chain
mc <- RWM(logpi = logpi_gauss, nits = nits, h = 2.2, x_curr = c(10,0))

# plot the chain and output the acceptance rate
plot(mc$x_store[,1:2], type = 'b', xlab = "X1", ylab = "X2")
```



There are many questions we can ask about the algorithm just mentioned. Here are some of them:

(1) Does the X_n generated do indeed have equilibrium distribution $\pi(\cdot)$?

(2) The acceptance rate of the algorithm changes as we change the tuning factor σ. What is an optimal acceptance rate?

(3) How well will the algorithm work in higher dimensional cases?

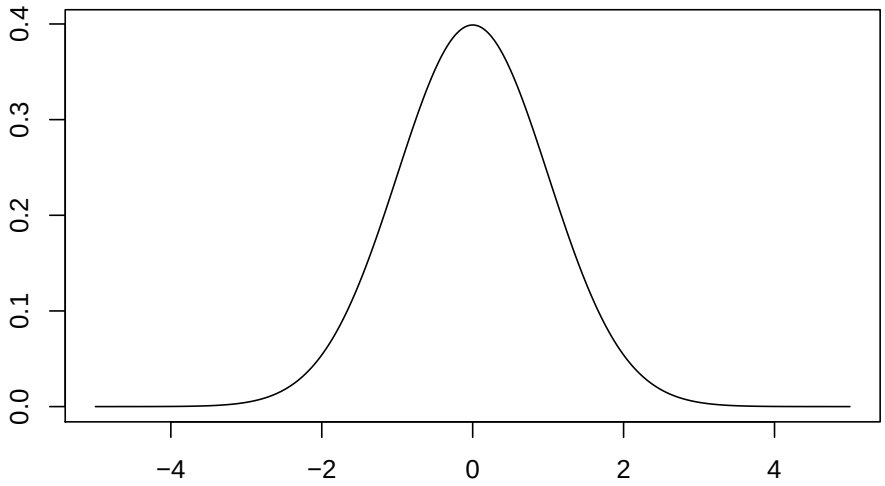
These three questions have all be answered in the literature. In this notes, we will not cover the technical parts of them and only highlight the results. For (1), it is true but I will skip the proof here. For (2), 23% is the optimal acceptance rate for high dimensional (≥ 5) cases (Gelman, Gilks, and Roberts 1997), while 44% is the optimal acceptance rate for dimension 1. The first quantity is obtained by studying the diffusion model which is the limit model of random walk as $d \to \infty$. However, a lot of other results are only obtained via numerical evidence. We need to, then, tune the factor such that the acceptance rate is close to the respective optimal value. For (3), in short, not too well. This is also a big motivation of researchers to develop other algorithms that work better in high dimension.

3 Hamiltonian Monte Carlo

The updates of Metropolis-Hastings is achieved by random walk dynamics, while it is adjusted by the acceptance function α. The updates of Hamiltonian Monte Carlo (HMC), on the other hand, is achieved by Hamiltonian dynamics, and it is also adjusted by the acceptance function. For MH, the target distribution is only used to adjust the output and does not affect the updates. For HMC, the information about the target distribution is affecting the updates, via $\nabla \log \pi(\cdot)$, which is related to the gradient of the target distribution. The gradient assisted updates are not only used by HMC, but also by other algorithms such as Metropolis-adjusted Langevin algorithm (MALA). MALA is very similar to MH. The proposal kernel of RWM is $N(X_i, \sigma^2 I)$ but the proposal kernel of MALA is $N(X_i + \frac{\sigma^2}{2}\Sigma\nabla \log \pi(X_i), \sigma^2\Sigma)$ of some Σ. One of the two changes is the additional drift $\frac{\sigma^2}{2}\Sigma\nabla \log \pi(X_i)$, which is where the information about the gradient is used.

Two questions. Why involving gradient is good? Why $\nabla \log \pi(\cdot)$ and not $\nabla\pi(\cdot)$?

To answer the first question, we first use the example of a standard normal distribution. The pdf is plotted below.



To sample from $N(0, 1)$, intuitively we should obtain most of the samples with values between, say, $[-2, 2]$, and a few of them outside that interval. Notice that the samples are updated from some starting point and go to another point after some motion. If you are at the tails, you would want to move towards the centre. If you are near the centre, you want to stay around the region. This is exactly what the gradient is used for. If you are at, say, 4, the gradient will be very negative so you will move to the left towards 0. Similar is true for any other points.

To answer the second question on why we use $\nabla \log \pi(\cdot)$ and not $\nabla\pi(\cdot)$, it is partly due to the fact that we may not have the exact target distribution, but only that up to a constant multiple. For example, in the Bayesian inference work, the posterior distribution is normally only computed up to a multiple. This issue is resolved by including the log, since $\nabla \log \pi(\cdot) = \pi'(\cdot)/\pi(\cdot)$ and the constant multiple will be cancelled here.

Now, let us study the Hamiltonian Monte Carlo. Note that the following are heavily adapted from Neal's review (Neal and others 2011).

3.1 Hamiltonian Dynamics and its Discretisation

A Hamiltonian system operates on a d-dimensional position vector q and a d-dimensional momentum vector p. So, the whole system, $H(q, p)$ is $2d$-dimensional. The system is dictated by Hamilton's equations. For each $i = 1, 2, \cdots, d$, we have

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}$$
$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}.$$

For Hamiltonian Monte Carlo, we would usually worth with Hamiltonian functions that can be written as

$$H(q, p) = U(q) + K(p)$$

where $U(q)$ is called the potential energy and $K(p)$ is called the kinetic energy normally defined as

$$K(p) = p^T M^{-1} p/2$$

. In the HMC setting, we would normally define the potential energy to be $-\log \pi(\cdot)$ of our target distribution π. The kinetic energy form mentioned earlier could be viewed as the minus of the log pdf of centred Gaussian distribution with covariance M. Using this construct, we could write the Hamilton's equations as

$$\frac{dq_i}{dt} = [M^{-1}p]_i$$
$$\frac{dp_i}{dt} = -\frac{\partial U}{\partial q_i}.$$

There are a lot of interesting and helpful properties about Hamiltonian dynamics that are essential to the theoretical justification of HMC, but we will not cover them here. Interested readers could take a look at Neal and others (2011).

The next thing that we will be learning is the discretising of the Hamilton equations. Since we are working with computers, discrete objects are what computers can comprehend, so we much make some approximation to our dynamical system. We will find a way to compute the position and momentum vectors at time ε, 2ε, 3ε, ... for some small ε.

The discretising method that we will use for HMC is called the Leapfrog method. It works as follow. For each $i = 1, 2, \cdots, d$ and at starting time t, we have

$$p_i(t + \varepsilon/2) = p_i(t) - (\varepsilon/2)\ \frac{\partial U}{\partial q_i}(q(t))$$
$$q_i(t + \varepsilon) = q_i(t) + \varepsilon\ \frac{p_i(t + \varepsilon/2)}{m_i}$$
$$p_i(t + \varepsilon) = p_i(t + \varepsilon/2) - (\varepsilon/2)\ \frac{\partial U}{\partial q_i}(q(t + \varepsilon)).$$

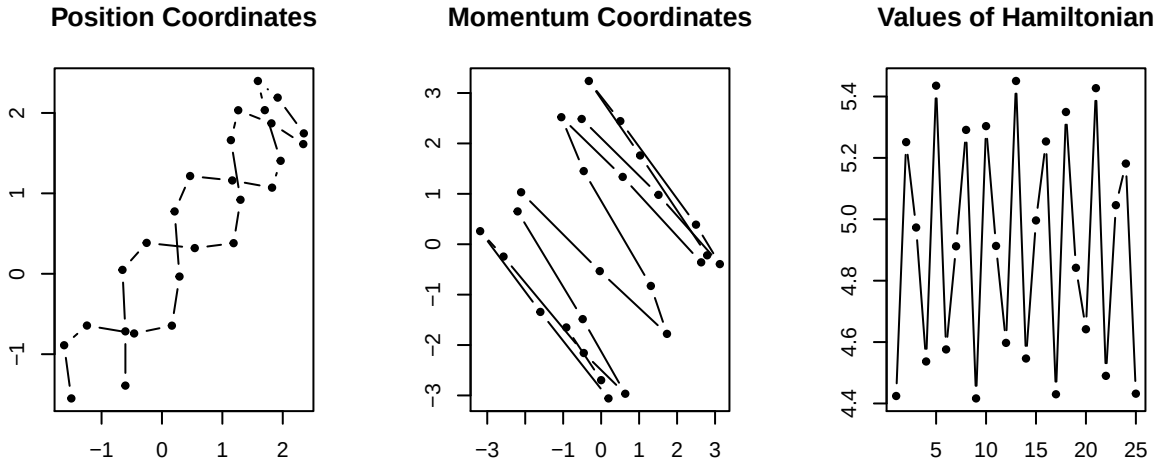
So, we will start with a half step for the momentum, then a full step for the position, and lastly the second half step for the momentum. This represents one step of the Leapfrog, and we will be doing a total of L step with stepsize ε.

To illustrate how the algorithm would work, we have the following plots of a trajectory of a 2d Gaussian simulated using Leapfrog($L = 25, \varepsilon = 0.25$). The starting position is $q = [-1.50, 1.55]^T$.

```
HMC_traj <- function(U, grad_U, eps, L, current_q){
  # Arguments:
  # U           Function that returns the potential energey given a value for position q
```

```r
  # grad_U      Vector of partial derivatives functions of U
  # eps         Stepsize of Leapfrog
  # L           Number of steps of Leapfrog
  # current_q   Current position q

  # KE is assumed to be sum(p_i^2)/2, with m_i = 1


  # step 1: draw momentum p from Gaussian
  q <- current_q
  p <- matrix(rnorm(length(q)),ncol=1)
  current_p <- p
  output_q <- c(q)
  output_p <- c(p)
  output_H <- c(U(q) + sum(p^2) / 2)

  # step 2: Leapfrog(L, eps) from (q,p)
  p <- p - eps / 2 * grad_U(q)
  for (i in 1:L){
    q <- q + eps*p
    output_q <- c(output_q,q)
    if (i!=L) {p <- p - eps*grad_U(q)/2
    output_H <- c(output_H, U(q) + sum(p^2) / 2)
    p <- p - eps*grad_U(q)/2
    output_p <- c(output_p,p)}
  }
  p <- p - eps / 2 * grad_U(q)
  output_p <- c(output_p,p)
  p <- -p

  # step 3: accept / reject change
  current_U <- U(current_q)
  current_K <- sum(current_p^2) / 2

  proposed_U <- U(q)
  proposed_K <- sum(p^2) / 2

  # if (runif(1) < exp(current_U + current_K - proposed_U - proposed_K)){
  #   return(q)
  # }
  # else{
  #   return (current_q)
  # }

  return(list(p = matrix(output_p,nrow=2),q = matrix(output_q,nrow=2), H = output_H))
}

set.seed(22)

sig <- matrix(c(1,0.95,0.95,1),nrow=2)
potential <- function(q){ t(q)%*%solve(sig)%*%q/2 }
grad_potential <- function(q){ t(t(q)%*%solve(sig)) }
```

```r
par(mfrow=c(1,3))

path <- HMC_traj(potential,grad_potential,0.25,25,current_q = matrix(c(-1.5,-1.55),ncol=1))

plot(path$q[1,],path$q[2,],type="b",main = "Position Coordinates",xlab="",ylab="",pch=20)
plot(path$p[1,],path$p[2,],type="b",main = "Momentum Coordinates",xlab="",ylab="",pch=20)
plot(path$H,type = "b",main = "Values of Hamiltonian",xlab="",ylab="",pch=20)
```



3.2 Hamiltonian Monte Carlo

The HMC algorithm is very similar to that of RWM, and the only difference is that the updates are derived by Leapfrog, and the acceptance function is adjusted.

Input: Starting value $X_0 = x$, target distribution $\pi(\cdot)$, number of iterations n, acceptance rate $\alpha(\cdot, \cdot)$, Leapfrog variables L and ε.

Main: For $i \in \{0, 1, \cdots, n\}$:

1. $Y = \text{Leapfrog}(L, \varepsilon)$ starting at X_n
2. Draw $U \sim U[0, 1]$,
3. If $U \leq \alpha(X_i, Y)$, let $X_{i+1} = Y$. Else, let $X_{i+1} = X_i$.

Ouput: $\{X_n\}$

Here the acceptance function is, for starting point (q, p) and proposed point $(q*, p*)$,

$$\min[1, \exp(H(q, p) - H(q*, p*))].$$

References

Gelman, A., W. R. Gilks, and G. O. Roberts. 1997. "Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms." *The Annals of Applied Probability* 7 (1). doi:10.1214/aoap/1034625254.

Hastings, W. K. 1970. "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika* 57 (1): 97–109. doi:10.1093/biomet/57.1.97.

Metropolis, Nicholas, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. 1953. "Equation of State Calculations by Fast Computing Machines." *The Journal of Chemical Physics* 21 (6): 1087–92. doi:10.1063/1.1699114.

Neal, Radford M, and others. 2011. "MCMC Using Hamiltonian Dynamics." *Handbook of Markov Chain Monte Carlo* 2 (11): 2.

Norris, J. R. 1998. *Markov Chains*. 1st pbk. ed. Cambridge Series on Statistical and Probabilistic Mathematics. Cambridge University Press.

Peskun, P. H. 1973. "Optimum Monte-Carlo Sampling Using Markov Chains." *Biometrika* 60 (3): 607–12. doi:10.1093/biomet/60.3.607.