Lecture 7

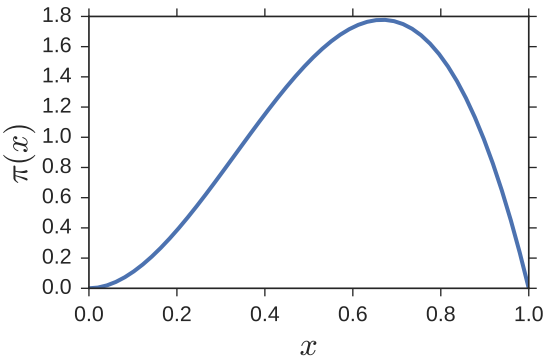
Metropolis Monte Carlo

Contents

1 Motivating Example

Consider sampling the following 1D distribution:

$$\pi(x) = 12x^2(1-x), \quad 0 \leq x < 1. \tag{1}$$



Due to its simplicity, it is probably wise to use direct sampling to sample this distribution.

1.1 Direct Sampling

(i) **Transformation Method**

If $u \sim U[0,1]$, then

$$u = F(x) = \int_0^x \pi(x')dx' = 4x^3 - 3x^4.$$

In principle, we could solve this quartic equation, and use the real root of x that lies in the domain $[0,1]$.

(ii) **Accept-Reject Method**

Alternatively, we could enclose this PDF within a box of height ≈ 2.0, and throw darts at it.

1.2 Objective

We shall consider how to apply an MCMC method to sample this distribution. We enter this exercise with low expectations; we expect this method to be inferior to direct sampling for simple problems.

Later, we shall consider problems for which MCMC is the better suited. However such problems tend to be complicated, and are perhaps less than ideal "first examples" to learn how MCMC works.

In some ways, this is similar to MC versus quadrature for integration. For 1D integrals quadrature methods work better. Likewise, for such simple $\pi(x)$ direct sampling methods work well, and should indeed be preferred.

2 Metropolis MCMC

Metropolis MCMC consists of two steps: (i) a *proposal* step, where a new "state" is suggested based on the current state, and (ii) an *acceptance* step, where the new state is either accepted or rejected.

2.1 Proposal

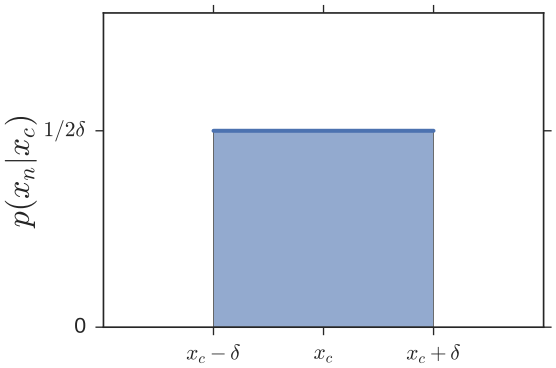
Consider a point $x_c \in [0, 1]$ in the domain of $\pi(x)$. We propose moving to a nearby point randomly using,[1]

$$x_n = x_c + \delta \times U[-1, 1], \tag{2}$$

where $U[-1, 1]$ represents a uniformly distributed random number between -1 and 1, and δ is a parameter that sets the size of jumps.

Formally, this proposal can be represented as a conditional probability,

$$W(x_c \to x_n) = p(x_n|x_c) = U[x_c - \delta, x_c + \delta] = \frac{1}{2\delta}. \tag{3}$$



2.2 Acceptance and Detailed Balance

If we want to sample the distribution $\pi(x)$, we can't simply accept all proposals. Like rejection sampling in direct MC, proposed moves in MCMC are accepted or rejected based on some criterion that ensures we sample the correct distribution.

Detailed balance provides the necessary guidance. Recall that detailed balance requires "net traffic" between any two states to be zero. For the two points x_c and x_n, this implies:

$$W(x_c \to x_n)\pi(x_c) = W(x_n \to x_c)\pi(x_n). \tag{4}$$

Transition probabilities W can be decomposed into the proposal p and acceptance a contributions:

$$W(x_c \to x_n) = p(x_c \to x_n)\, a(x_c \to x_n). \tag{5}$$

In Metropolis MCMC, the proposal functions are symmetric. For our particular example,

$$\begin{aligned} p(x_c \to x_n) &= U[x_c - \delta, x_c + \delta] = \frac{1}{2\delta}, \\ p(x_n \to x_c) &= U[x_n - \delta, x_n + \delta] = \frac{1}{2\delta}. \end{aligned} \tag{6}$$

Thus, the probability of proposing x_n when the current state is x_c is the same as the reverse step.

$$p(x_c \to x_n) = p(x_n \to x_c) = \frac{1}{2\delta}.$$

[1] The subscript "c" and "n" stand for "current" and "next".

Using eqns (5) and (6) in (4),

$$a(x_c \to x_n)\pi(x_c) = a(x_n \to x_c)\pi(x_n), \tag{7}$$

$$\frac{a(x_c \to x_n)}{a(x_n \to x_c)} = \frac{\pi(x_n)}{\pi(x_c)}. \tag{8}$$

Metropolis et al. suggested that *one way of satisfying* detailed balance in eqn (8) was to choose,

$$a(x_c \to x_n) = \begin{cases} 1, & \text{if } \pi(x_n) > \pi(x_c) \\ \frac{\pi(x_n)}{\pi(x_c)}, & \text{otherwise.} \end{cases} \tag{9}$$

Or in shorthand,

$$a(x_c \to x_n) = \min\left\{1, \frac{\pi(x_n)}{\pi(x_c)}\right\}. \tag{10}$$

How in the world is eqn (9) consistent with detailed balance, eqn (4)?

To see why this choice works, consider the two possible cases,

(i) $\pi(x_n) \geq \pi(x_c)$

In this case, the proposed state is more probable than the current state. Using eqn (9),

$$a(x_c \to x_n) = 1, \text{ and}$$
$$a(x_n \to x_c) = \pi(x_c)/\pi(x_n).$$

Thus, moves to x_n are always selected, while the reverse is sometimes rejected. The frequency of rejection depends on the ratio $\pi(x_c)/\pi(x_n)$. Taking their ratio,

$$\frac{a(x_c \to x_n)}{a(x_n \to x_c)} = \frac{1}{\pi(x_c)/\pi(x_n)} = \frac{\pi(x_n)}{\pi(x_c)},$$

as required by detailed balance (eqn. 8).

(ii) $\pi(x_n) < \pi(x_c)$

We can analyze this case similarly. Using eqn (9),

$$a(x_c \to x_n) = \pi(x_n)/\pi(x_c)$$
$$a(x_n \to x_c) = 1.$$

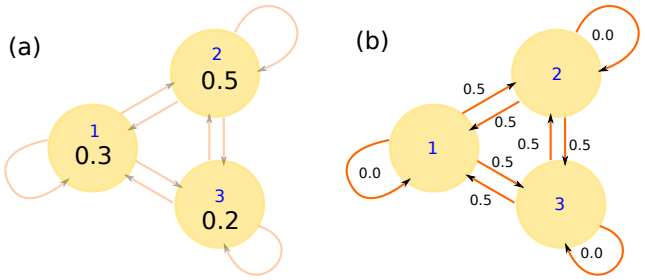
Taking the ratio,

$$\frac{a(x_c \to x_n)}{a(x_n \to x_c)} = \frac{\pi(x_n)}{\pi(x_c)}.$$

Notation Warning: Here we expanded the transition from state x_c to state x_n as $x_c \to x_n$, so that there was no notational confusion. Remember that people often write $p(x_c \to x_n)$ as $p(x_n|x_c)$, or even p_{nc}.[2]

Example 1: Consider a system with three states labeled 1, 2, and 3 with $\pi_1 = 0.3$, $\pi_2 = 0.5$ and $\pi_3 = 0.2$, shown in the figure below.

[2]To make matters worse some people also write p_{cn} to mean $p(x_c \to x_n)$. It can cause a lot of unnecessary confusion. We will stick with our standard notation, where $W(j \to i) = W_{ij}$, etc.



Suppose we design a symmetric proposal move as shown in subfigure (b), where $p_{ij} = 0.5$ for $i \neq j$, and $p_{ii} = 0$. Using Metropolis acceptance criteria (eqn 9) determine the acceptance probabilities a_{ij} for all $i \neq j$.[3]

Solution: Let's start with a_{21} and a_{12}. Since $\pi_2 > \pi_1$, $a_{21} = 1$, while $a_{12} = \pi_1/\pi_2 = 0.3/0.5 = 0.6$. Similarly, $a_{23} = 1$ and $a_{32} = 0.2/0.5 = 0.4$, while $a_{13} = 1$ and $a_{31} = 0.2/0.3 = 0.67$.

2.3 Transition Probability Matrix

What transition probability matrix (TPM) does Metropolis MCMC lead to? This is only a theoretical concern; practically we don't have to deal with the TPM $\mathbf{W}$ in its entirety.

In Metropolis MCMC, a move from state j to i is proposed with probability p_{ij}. It may be accepted with probability a_{ij} given by the Metropolis criterion, or it may be rejected, in which case $i = j$.

Rejection affects only "diagonal" terms of $\mathbf{W}$. For "off-diagonal" terms ($j \neq i$), $W_{ij} = p_{ij}a_{ij}$ is the product of the proposal and acceptance probabilities.

A diagonal term can be non-zero even if "self-moves" are not explicitly proposed. That is $p_{jj} = 0$ does not imply $W_{jj} = 0$, since it accumulates weight due to rejection of proposed moves. Formally,

$$W_{ij} = p_{ij}a_{ij} + \delta_{ij} \sum_{i'} p_{i'j}(1 - a_{i'j}), \tag{11}$$

δ_{ij} is the Kronecker delta function. It ensures that the second term, which captures the effect of rejections, is activated only for diagonal terms, where $i = j$.

Let us ponder over this second term. The summation here extends over all possible states i' that are accessible from state j. Note that $p_{i'j}(1 - a_{i'j})$ is the probability that a move proposed to state i' is rejected.[4]

Note that the TPM depends on the choice of proposal and acceptance functions p_{ij} and a_{ij}. Furthermore, Metropolis MCMC does not require the target distribution $\pi(x)$ to be normalized, since it only deals with ratios. This is a big deal!

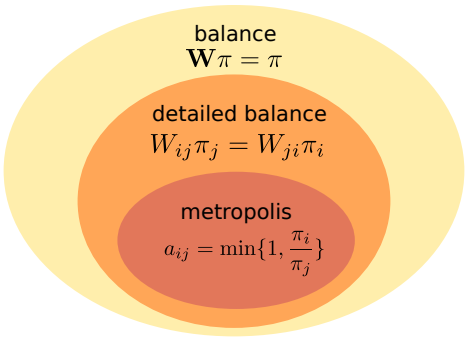
2.4 Big Picture

It is worthwhile to step back and look at the conceptual link between Markov chains (characterized by TPM $\mathbf{W}$) and the Metropolis acceptance criteria (see picture below).

Recall that our primary goal is to sample $x \sim \pi(x)$. In MCMC, the first step is to design an appropriate $\mathbf{W}$.

[3] Note that a_{ii} is meaningless, since $p_{ii} = 0$. You can't accept or reject a move you never propose!

[4] You are strongly encouraged to try Thought Question (i) which illuminates this idea for the Example above.



The condition of **balance** is mandatory for all transition probability matrices. This requirement ensures that regardless of where we start, after sufficient number of steps, we sample the target distribution.

However, as we said before, the design problem is underspecified, and we are free to make a few design choices. Said differently, we can self-impose additional constraints, if we find them to be helpful. One such constraint that we willingly and frequently (almost always) impose is **detailed balance**. Recall it is a stricter condition than "balance".

Metropolis MCMC is one further step removed. It is a particular choice/algorithm for implementing detailed balance. There are other choices, including, for example, Glauber "dynamics"

$$a(c \to n) = \frac{\pi_n}{\pi_c + \pi_n} = \frac{1}{1 + \pi_c/\pi_n} \tag{12}$$

Exercise: Show that this choice also satisfies detailed balance, eqn. 8.

Exercise: If $\pi_n/\pi_c = 1$, what is $a(c \to n)$ according to Metropolis and Glauber?

3 Simulation

A typical MCMC code has the following subroutines:

(i) the distribution $\pi(x)$

(ii) proposal function, which draws $x_n \sim p(x_n|x_c)$

(iii) acceptance function, which applies the Metropolis criterion

(iv) a driver, which initializes and orchestrates computation

The design is modular. If we want to test a different proposal or acceptance criterion, then we can leave the rest of the machinery in place, and only replace or alter the relevant function.

Let us illustrate this for the 1D example considered at the beginning of this chapter,

$$\pi(x) = 12x^2(1-x), \quad 0 \le x < 1.$$

We sacrifice efficiency for clarity in the subroutines listed below.

3.1 Probability Distribution

This function takes in a state x, and returns the target probability $\pi(x)$.

```
def probdist(x):
    if x < 0. or x > 1.:
        return 0.
    else:
        return 12*x**2*(1-x)
```

Note that we explicitly treat cases for x outside the domain $0 < x < 1$ because it is possible that some proposals take us outside the prescribed range.

3.2 Proposal Function

This function accepts a user-specified maximum step-size δ, and the current state to propose a new state x_n.

```
def proposal(xc, delta):
    xn = np.random.uniform(xc-delta, xc+delta)
    return xn
```

Note that this might result in x_n outside the "domain", i.e. it is possible that $x_n > 1$ or $x_n < 0$.

3.3 Acceptance function

This function looks at the ratio of the probabilities $\pi(x_n)/\pi(x_c)$, and applies the Metropolis acceptance criterion to determine if the proposed move should be accepted.

```
def metropolis_accept(xc, xn, f):
    acc   = False
    ratio = f(xn)/f(xc)

    if ratio > 1.:
        acc = True
    elif np.random.rand() < ratio:
        acc = True
    return acc
```

If $\pi(x_n)/\pi(x_c) < 1$, then the acceptance probability according to eqn (9) is $\pi(x_n)/\pi(x_c)$. If the ratio is large, say 0.9, then we should accept the proposed move 90% of the time. This can be accomplished by generating a uniform random number between 0 and 1, and accepting the move if this random number is less than $\pi(x_n)/\pi(x_c)$. [5]

3.4 Driver

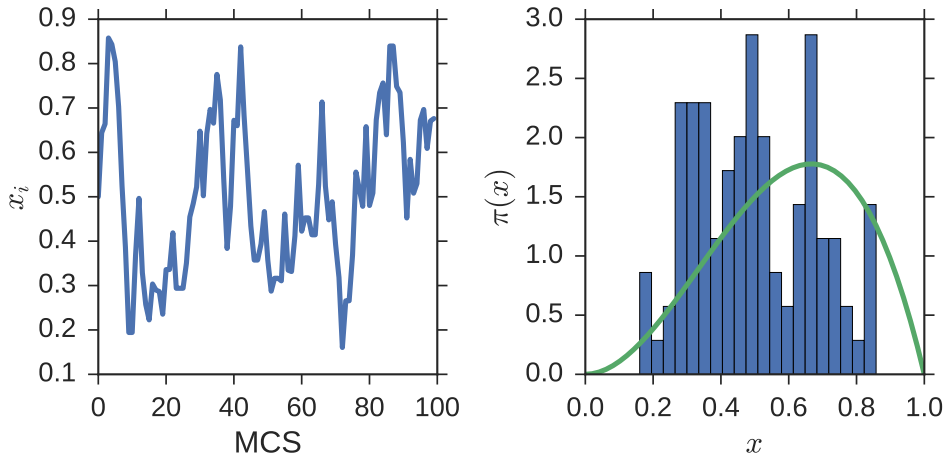
Finally we tie these three functions using a driver routine. The driver takes three inputs: the number of Monte Carlo steps (MCS), the maximum step size δ for the proposal move, and an initial state x_0.

[5] This is similar to modeling a unfair coin with a bias for, say heads, with a probability $\pi(x_n)/\pi(x_c)$.

```
def metropolisMCMC(nmcs, delta, x0):
    # initialize
    x     = np.zeros((nmcs)) # samples
    xc    = x0               # initial state
    nSucc = 0                # #accept

    # main loop of proposal/rejection
    for imcs in range(nmcs):

        x[imcs] = xc
        xn      = proposal(xc, delta)

        if(metropolis_accept(xc, xn, probdist)):
            xc = xn
            nSucc += 1

    return x, float(nSucc)/nmcs
```

There are a lot of subtleties in implementing Metropolis MCMC. For now let us just run the program, and observe the output. Let us start with a relatively short simulation of 100 MCS, with $\delta = 0.2$ and $x_0 = 0.5$ smack in the middle of the domain.

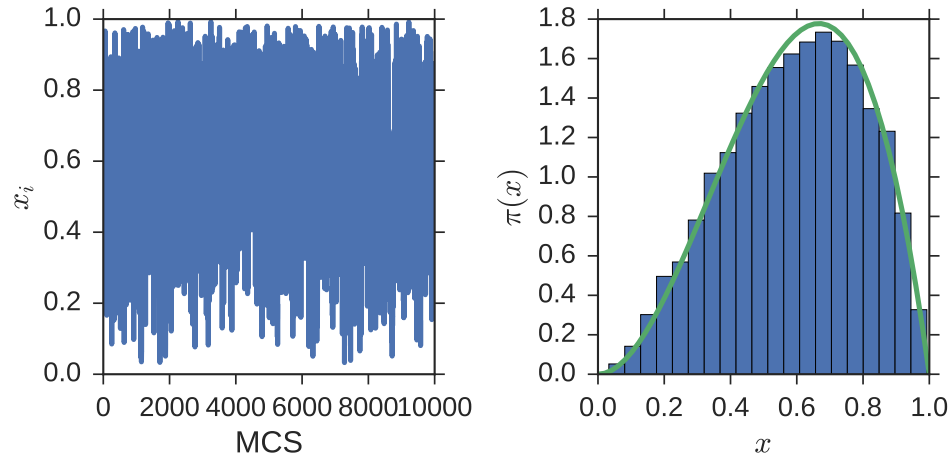
```
x, succRatio = metropolisMCMC(100, 0.2, 0.5)
```



We obtain a **traceplot** shown on the left, as x samples the domain between 0 and 1. The acceptance ratio is 0.83. The figure on the right compares the histogram obtained from the x sampled and the target PDF.

As we increase the number of MCS, the correspondence between the histogram and $\pi(x)$ improves. The acceptance ratio remains close to 0.83.

```
x, succRatio = metropolisMCMC(10000, 0.2, 0.5)
```



4 Issues

We wrap up this chapter by bookmarking future points of discussion. But first, we note that Metropolis MCMC is extremely powerful, versatile, and ubiquitous. For many, relatively benign problems, this is all the MCMC you will ever need to know!

Even so, it may have been obvious to you that we swept several issues under the carpet. These include:

(i) **Proposal**: We need to specify proposal $p_{nc} = \pi(x_n|x_c)$. How do we pick the form of the proposal? Here we picked a uniform distribution, but we could also pick a normal distribution.

As we shall see later, we don't have to stick with symmetric proposal moves. *Metropolis-Hastings* is a generalization of Metropolis, which allows us to break this requirement, by suitably modifying the acceptance criterion.

Furthermore, we need to set a maximum or characteristic step size δ. If δ is too small, the Markov chain travels extremely slowly.[6] However if δ is too large the acceptance ratio drops, and most moves are rejected. Such a choice may be too aggressive.

(ii) **Correlation**: Unlike direct MC, the sequence of x_i generated are not independent. This correlation is visible in traceplots. As we have foreshadowed throughout the class, this complicates analysis. A practical approach to addressing uncertainty that we shall adopt is called **block averaging**.

(iii) **Burn-in**: The first few samples x_i are typically tainted by the choice of x_0. Often, we discard samples generated during this so-called "burn-in" phase. We start the "production run" after this initialization phase is complete.[7] We start analyzing samples after Markov chain is stationary. Note that none of this business was required in direct MC.

(iv) **Length of simulation**: How long do we run the MCMC simulation? How do we find out if our simulation has "converged"? Mathematically, convergence is governed by the magnitude

[6] From a performance standpoint, such a conservative choice are similar to choosing a small timestep in solving ODEs.

[7] In our toy example, we did not do this. However, our initial state of $x_0 = 0.5$ was well-chosen, since it was near the peak of the target distribution.

of the second largest eigenvalue of $\mathbf{W}$. In practice this is a hard question with many answers; but none are completely bullet-proof.

Hopefully, by the time we finish the course, you will come to understand and respect some of these issues. In addition, we shall keep adding to our suite of algorithms for MCMC.

4.1 Three-Peaks Example

So what does a good MCMC algorithm or calculation look like?

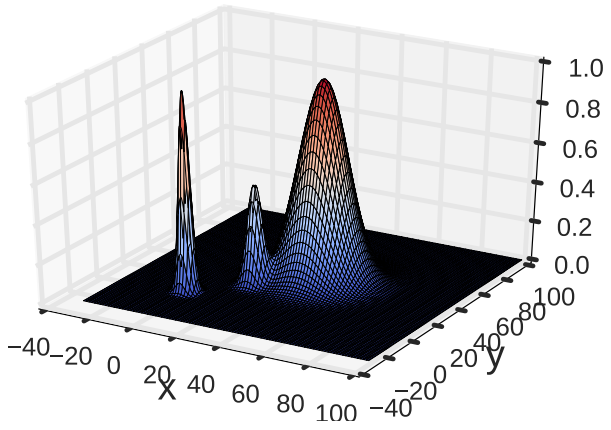
- It converges in a reasonable amount of time
- It samples all relevant parts of the domain
- It does not get stuck in local "peaks"

Often this requires us to tune the parameters of the algorithm.

Next let us consider a 2D "three-peaks" problem, which allows us to explore some of these issues (credit: Peter Beerli). This is a harder problem than the simple 1D problem we saw so far. The unnormalized PDF is given by,

$$\begin{aligned}\pi(x,y) \sim &\exp\left(-\frac{1}{10}(x^2+y^2)\right) + \\ &\frac{1}{2}\exp\left(-\frac{1}{20}((x-20.)^2+(y-20.)^2)\right) + \\ &\exp\left(-\frac{1}{200}((x-40.)^2+(y-40.)^2)\right)\end{aligned} \tag{13}$$

defined over a finite domain $\Omega = [-30, 100]^2$.

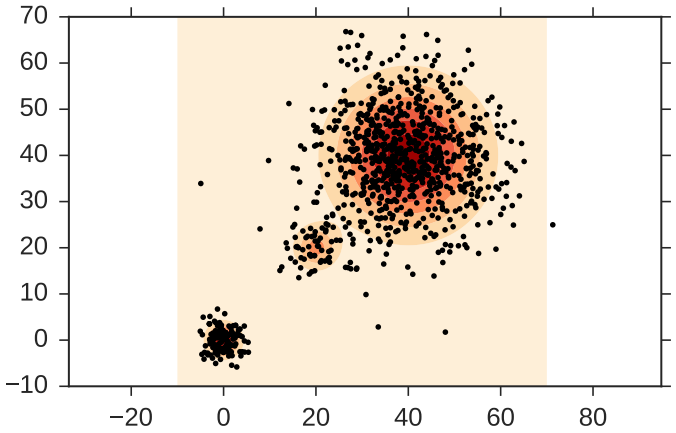


There is a sharp peak at (0,0), a less sharp peak at (20,20) and a diffuse peak at (40,40).

We want to use Metropolis MCMC to sample this distribution. The code, as usual, includes four pieces: (i) the probability distribution,[8] (ii) the proposal, (iii) the acceptance criterion, and (iv) driver. They are listed in the appendix in section A.1.1

[8] Note that we don't need to explicitly know the normalization constant.

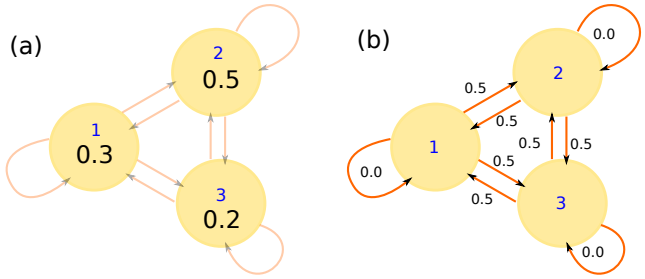
A typical result is plotted below. Experiment with the code and the settings. Try different values of δ from small to large. Pick different starting points. For now, do not worry too much about accuracy; just making sure that all the peaks are sampled is non-trivial.



5 Problems

5.1 Thought Questions

(i) Consider the 3 state example considered in the solved example.



(a) Use eqn. (11) to write down the TPM $\mathbf{W}$.

(b) Verify that $\mathbf{W}$ obey the conditions of normalization and balance.

(ii) Plot the acceptance rate $a(c \to n)$ as a function of π_n/π_c for Metropolis and Glauber acceptance criteria.

5.2 Numerical Problems

(i) Mixture of Gaussians

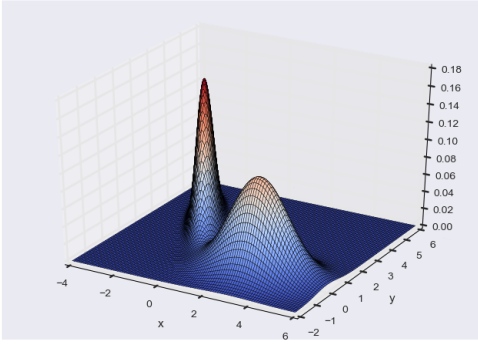
Consider the following mixture of 2D Gaussian distributions,

$$\pi(x, y) = 0.6\mathcal{N}(\mu_1, \Sigma_1) + 0.4\mathcal{N}(\mu_2, \Sigma_2),$$

where,

$$\mu_1 = \begin{bmatrix} 3 \\ 0 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} 1.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \text{and}$$

$$\mu_2 = \begin{bmatrix} -1.25 \\ 2.5 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 0.5 & -0.6 \\ -0.6 & 1.0 \end{bmatrix}.$$



This distribution is shown in the figure on the right. Holding all other parameters constant, vary the following one at a time:

(a) number of MC steps

(b) the choice of δ

(c) use a normal proposal function instead of uniform

(d) initial state

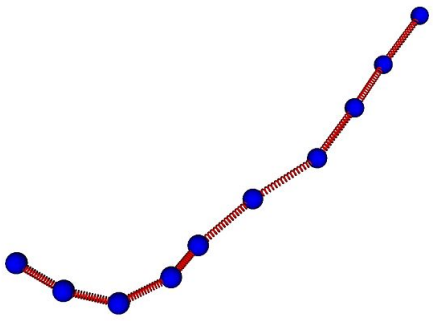
Tune the parameters, until you are "visually" satisfied with the sampling. Report your observations, and try to explain them in terms of (a) the acceptance ratio, and (b) peaks exploration of both peaks?

(ii) Bead-Spring Polymer Model

A popular molecular model for polymers is the bead-spring model. Consider a polymer comprised on N atoms, as shown in the picture below.

The location of each atom is given by $\boldsymbol{r}_i$. The energy corresponding to a particular configuration is given by:

$$U(\{\boldsymbol{r}^N\}) = U_{LJ} + U_{FENE}.$$



The repulsive "Lennard-Jones" force between all pairs of atoms is given by,

$$U_{LJ} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} u_{LJ}(r_{ij}),$$

where, $r_{ij} = |\boldsymbol{r}_i - \boldsymbol{r}_j|$ is the distance between atoms i and j, and

$$u_{LJ}(r) = \begin{cases} 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] + \epsilon, & r < 2^{1/6}\sigma \\ 0, & \text{otherwise} \end{cases}$$

The attractive spring potential is given by,

$$U_{FENE} = -0.5KR_0^2 \sum_{i=1}^{N-1} \ln\left[1 - \left(\frac{r_{i,i+1}}{R_0}\right)^2\right],$$

where $r_{i,i+1} = |\boldsymbol{r}_{i+1} - \boldsymbol{r}_i|$.

For this problem, suppose $\sigma = 1.0$, $\epsilon = 1.0$, $K = 30$, and $R_0 = 1.5$.

Compute the mean end-to-end distance $R_e = \boldsymbol{r}_N - \boldsymbol{r}_1$ given by:

$$\langle R_e^2 \rangle = \frac{\int R_e^2(\boldsymbol{r}^N) e^{-U(\boldsymbol{r}^N)} d\boldsymbol{r}^N}{\int e^{-U(\boldsymbol{r}^N)} d\boldsymbol{r}^N}.$$

A Appendix

A.1 Python Code

A.1.1 Code for Three-Peaks

The four functions used to perform MCMC calculations on the three-peaks example (sec 4.1) are listed here. We start with a function that defines the unnormalized probability distribution $f(x, y)$.

```
def probdist(x, y, domain):
        if(domain[0][0] < x < domain[0][1]) and (domain[1][0] < y < domain[1][1]):
                f  = np.exp(-1/10. * (x**2 + y**2))
                f += 0.5 * np.exp(-1/20. * ((x-20.)**2 + (y-20.)**2))
                f += np.exp(-1/200. * ((x-40.)**2 + (y-40.)**2))
        else:
                f  = 0.
        return f
```

The proposal function, which again, uses a uniformly distributed step of maximum step-size δ.

$$(x_n, y_n) = (U[x_c - \delta, x_c + \delta], U[y_c - \delta, y_c + \delta])$$

```
def proposal(x, y, delta):
        newx = x + delta * (2. * np.random.rand() - 1.)
        newy = y + delta * (2. * np.random.rand() - 1.)
        return newx, newy
```

The acceptance function is the standard Metropolis criterion.

```
def metropolis_accept(newx, newy, oldx, oldy, domain):
    """acceptance function"""

    numer = probdist(newx, newy, domain);
    denom = probdist(oldx, oldy, domain);

    if np.random.rand() < numer/denom:
        accept = True
        x, y = newx, newy
    else:
```

```
        accept = False
        x, y = oldx, oldy

    return accept, x, y
```

Finally, a driver routine to tie all of them together.

```
def driver(nsteps=10000, delta=10., thin=10, x0 = 0.0, y0 = 0.0):
        """
                nsteps = 10000 # number of steps in the mcmc chain
                delta  = 10.   # delta for the proposal
                thin   = 10    # save every 10th point
                x0, y0 = (0,0) # initial stte
    """
        # preliminaries;

        domain = np.array([[-30.,100.],[-30.,100.]])

        x = x0; y = y0

        AccRatio = 0.  # target around 0.4
        NumSucc  = 0   # number of successes

        # record of points sampled after thinning
        recz = np.zeros((int(nsteps/thin), 2))

        for iMCS in range(nsteps): # Main Loop
                # propose -> accept
                newx, newy = proposal(x, y, delta)
                accept, newx, newy = metropolis_accept(newx, newy, x, y, domain)
                if accept:
                        NumSucc += 1
                        x, y = newx, newy

                if (iMCS % thin) == 0: # record
                        recz[int(iMCS/thin)] = [x, y];

        AccRatio = float(NumSucc)/float(nsteps)

        return recz, AccRatio
```

Finally, we call the function and plot the results

```
recz, acc = driver(nsteps=10000, delta=10., thin=10, x0 = 0.0, y0 = 0.0)
print(acc)

# Plotting
plt.plot(recz[:,0],recz[:,1],'k.')

x = np.linspace(-10,70,100)
X, Y = np.meshgrid(x,x)
m, n = X.shape
```

```
Z = np.zeros((m,n))


domain = np.array([[-30.,100.],[-30.,100.]])
for i in range(m):
        for j in range(n):
                Z[i,j] = probdist(X[i,j], Y[i,j], domain)

from matplotlib import cm
plt.contourf(X,Y,Z, cmap=cm.OrRd)
plt.axis('equal')
```