Markov Chain Monte Carlo

Metropolis-coupled MCMC or MC^3

Sachin Shanbhag

Department of Scientific Computing
Florida State University,
Tallahassee, FL 32306.

Background

In this lecture, we will consider Metropolis-coupled Markov Chain Monte Carlo, abbreviated as MCMCMC or MC^3.

It is an algorithm that has been “rediscovered” several times; each time it has been given a slightly different name:

- parallel tempering
- simulated tempering
- replica exchange Monte Carlo
- etc.

Furthermore, it is related to methods such as umbrella sampling, multi-canonical ensembles, and simulated annealing.

Background

The primary use of MC^3 is to improve mixing, by simultaneously running multiple Markov chains at different temperatures.

The label temperature has a meaning similar to that in simulated annealing.

Periodically, some of these chains are swapped.

It can be extremely useful for multimodal distributions in high-dimensional spaces.

Let's use a simple 1D bimodal distribution to motivate and illustrate this concept

Motivating Example

Consider a double-well potential:

$$U(x) = \gamma(x^2 - 1)^2, \qquad \gamma > 0$$

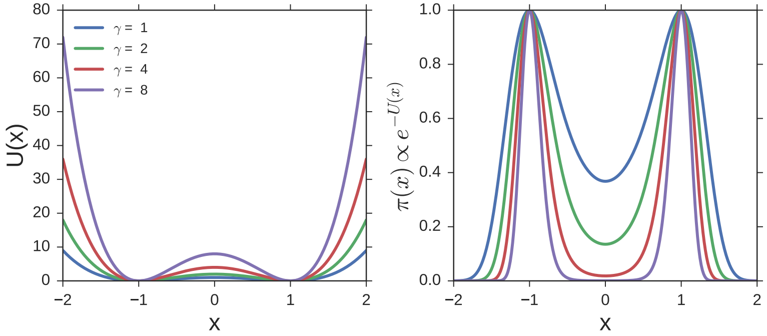
and a corresponding probability distribution:

$$\pi(x) \propto \exp(-U(x)).$$

```python
def U(x, gamma):
    return gamma * (x**2-1)**2

def pU(x, gamma):
    return np.exp(-U(x,gamma))
```

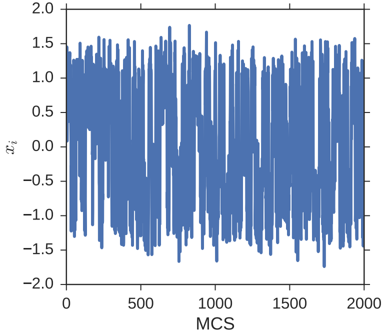
Example

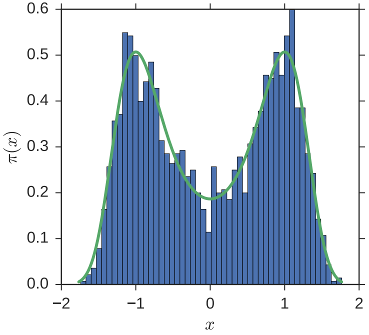


As γ increases from 1 to 8, we note that the bimodal peaks in $\pi(x)$ become sharper, and the "canyon" between the two peaks becomes deeper.

Let's try to run a simple Metropolis MCMC on this problem.

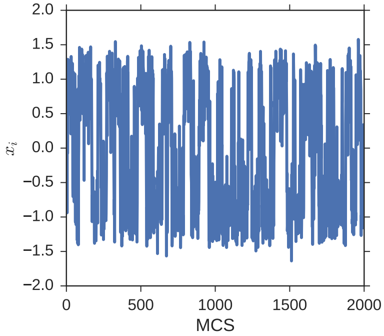
Results: $\gamma = 1$

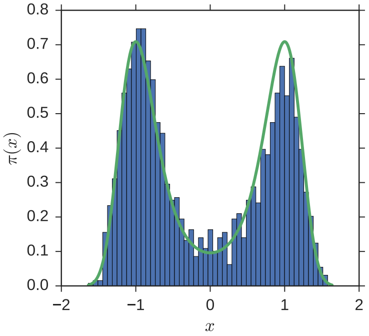




Acceptance Ratio $= 0.8467$

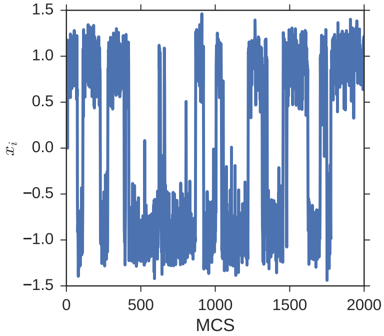
Results: $\gamma = 2$

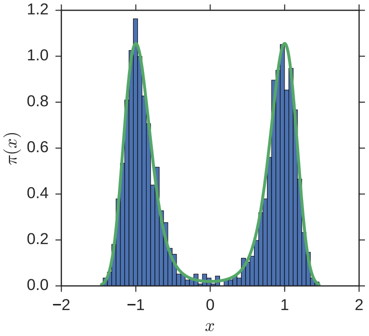




Acceptance Ratio $= 0.7646$

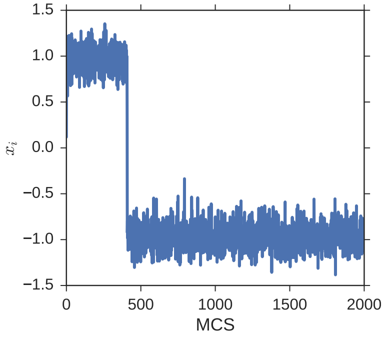
Results: $\gamma = 4$

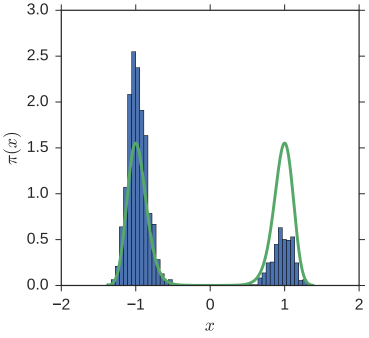




Acceptance Ratio $= 0.6489$

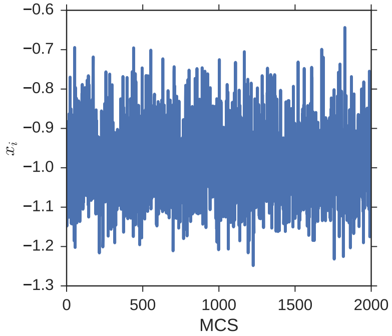
Results: $\gamma = 8$

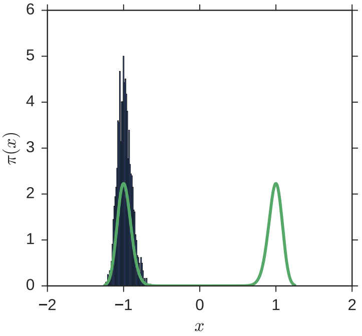




Acceptance Ratio = 0.5141

Results: $\gamma = 16$





Acceptance Ratio $= 0.4024$

Mixing

As γ increases the two peaks become well separated, and it becomes increasingly hard to travel from one to the other.

In this simple 1D example, there are many possible ways to counter this:

- take larger steps
- run a much much much longer simulation
- use multiple starting points, and average over them

In complex high dimensional problems, it may not be easy to visualize what is happening.

Hence, algorithms that can lead to better mixing are welcome.

MC^3: Background

The key idea is to run several chains at different temperatures.

Inspiration from thermodynamics; the Boltzmann factor $U(x)/T$ makes it easier for systems to jump over barriers at higher temperatures.

Consider the double-well potential at different temperatures T between 1 and ∞.

$$\frac{U(x)}{T} = \frac{\gamma}{T}(x^2-1)^2,$$

which corresponds to the probability distribution:

$$\pi(x,T) \propto \exp\left(-\frac{U(x)}{T}\right) = \left(e^{-U(x)}\right)^{1/T}.$$

Background

In the non-physics community it is common to write this as,

$$h(x, a) \propto (\exp -U(x))^{a} = (\pi(x))^{a},$$

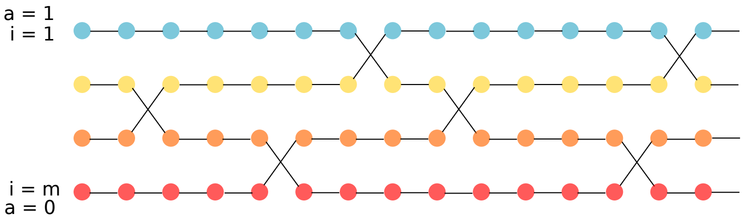
where, $h(x, a)$ defines a family of related PDFs, and a is understood to be the inverse temperature $1/T$.

As T varies between 1 and ∞, a varies between 1 and 0.

The different chains correspond to $h(x, a)$ at different values of a ranging between "1" (the cold chain), and "0" (the hot chain).

The exact number and temperatures can be optimized for particular problems, but here let us consider a simpler case, where we choose $m = 4$ chains, with $a_i = 1, 0.5, 0.2,$ and 0.01.

Schema



Note that we really only care about sampling the coldest chain (the target distribution).

The other chains are just “helpers” which help the cold chain from getting trapped.

Intuition

The intuition in some ways is similar to the intuition in simulated annealing.

For small a (corresponding to high temperatures), the landscape is flattened, allowing the hot chain to jump over "barriers".

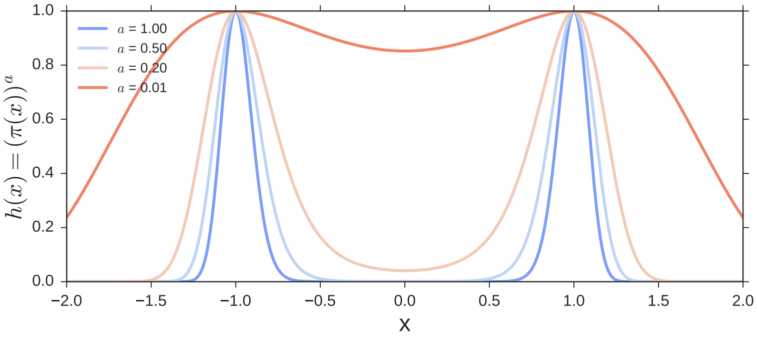
By swapping chains, we give the cold chain a chance to jump over the same barriers in a reasonable amount of time.

Specific Example

Let $h_i = (\pi(x))^{a_i}$, where $a_i = 1.0, 0.5, 0.2,$ and 0.01.

We can plot what the potential landscape (or probability distribution landscape) looks like to the different chains for $\gamma = 16$.

Landscape and Temperature



The cold chain has well-separated peaks.

The family of chains soften the barrier between the peaks.

Algorithm

Let me first present the algorithm, and then ponder over the swapping moves.

1. Initialize the m chains at the different temperatures
2. At each timestep update all the chains independently using a regular MH proposal.
3. Every few steps (say 10 MCS)
    - propose a swap between a randomly selected chain i and one of its neighbors $j = i \pm 1$.
    - accept with Metropolis criterion; the Hastings ratio is:

      $$r = \frac{h_i(x_j)h_j(x_i)}{h_i(x_i)h_j(x_j)},$$

      where x_i and x_j are the states of chains i and j, before the proposed swap.

      $$a_{ji} = \min\{1, r\}$$
4. Repeat steps 2 and 3

Rationale

How is the Hastings ratio for the swap determined?

Consider a symmetric swap proposal, so that $q_{ij} = q_{ji} = 1/2$.

For the end chains ($i = 1$ and $i = m$), it means we reject the swap if $j = 0$ or $j = m + 1$ is selected.

The idea behind MC3 is to think of a grand Markov process which consists of several independent Markov chains.

For illustration, suppose we have two chains with distributions h_1 and h_2; with "current states" x and y, respectively.

The "grand" Markov process then considers the stationary distribution corresponding to:

$$\pi(x, y) = h_1(x)h_2(y). \tag{1}$$

Rationale

For a symmetric proposal, $(x, y) \rightarrow (x^*, y^*)$, the Hastings ratio is:

$$r = \frac{\pi(x^*, y^*)}{\pi(x, y)} \tag{2}$$

Now consider the special proposal of swapping the two chains; i.e. $x^* = y$, and $y^* = x$. Then,

$$\begin{aligned} r &= \frac{\pi(x^*, y^*)}{\pi(x, y)} \\ &= \frac{h_1(x^*)h_2(y^*)}{h_1(x)h_2(y)} \qquad (3) \\ &= \frac{h_1(y)h_2(x)}{h_1(x)h_2(y)} \qquad (4) \end{aligned}$$

Application

Let us now write functions to implement MCMCMC for the two-well potential considered above, with $\gamma = 16$.

We've already have the function for the unnormalized PDF.

```
def pU(x, gamma):
    return np.exp(-gamma * (x**2-1)**2)
```

Note that since all the swap proposals are Metropolis-like, we don't need the normalization factor.

Furthermore, since we tried to use simple Metropolis we already have the proposal and accept functions.

```
def metro_proposal(xc, delta):
    xn = np.random.normal(xc, delta)
    return xn
```

Application

The Metropolis acceptance protocol for “intra-chain” moves is slightly modified to account for the $a = 1/T$ factor.

```python
def metro_accept(xn, pc, f, gamma, invTemp):
    """acceptance for within-chain proposals;
    xn = proposal; fc = current probability
    f, gamma = function/parameter to be sampled
    the factor a = 1/T = invTemp"""
    acc = False
    pn  = f(xn, gamma)
    ratio = (pn/pc)**invTemp # note change

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

Inter-chain Swaps

Let the inverse temperatures of the different chains be

$$\mathbf{a} = (a_1, a_2, ..., a_m).$$

We can represent the current state of the chains and corresponding probabilities by

$$\mathbf{z} = (x_1, x_2, ..., x_m),$$

$$\mathbf{h}(\mathbf{z}) = (h_1, h_2, ..., h_m),$$

where $h_i(x_i) = (\pi(x_i))^{a_i}$.

Consider the Hastings ratio for swapping two chains i and j:

Inter-chain Swaps

$$
\begin{aligned}
r &= \frac{h_i(x_j)h_j(x_i)}{h_i(x_i)h_j(x_j)} \\
&= \frac{\pi(x_j)^{a_i}}{\pi(x_i)^{a_i}} \frac{\pi(x_i)^{a_j}}{\pi(x_j)^{a_j}} \\
&= \left(\frac{\pi(x_j)}{\pi(x_i)}\right)^{a_i} \left(\frac{\pi(x_i)}{\pi(x_j)}\right)^{a_j} \\
&= \left(\pi(x_j)\right)^{a_i-a_j} \left(\pi(x_i)\right)^{a_j-a_i} \qquad (5)
\end{aligned}
$$

We write a function:

- to propose a swap between a randomly chosen chain, and one of its neighbors, and
- to accept or reject the proposed swap.

```python
def proposeSwap(nchain):
    """pick chains i and j to swap"""

    i = np.random.randint(nchain) # pick randomly

    # pick a neighbor
    if np.random.rand() < 0.5:
        j = i + 1
    else:
        j = i - 1

    # i = 0 or i = nchain - special consideration
    if j < 0:
        j = 0
    elif j > nchain-1:
        j = nchain-1

    return i, j
```

```python
def swap_accept(hi, hj, ai, aj):
    """should I accept the swap between chain i
    and j? Need function values and invTemp.
   Apply derived formula."""

    factor = hj**(ai-aj) * hi**(aj-ai)

    if np.random.rand() <= factor:
        swap = True
    else:
        swap = False

    return swap
```

Overall Program Structure

- Loop over `nsteps`
  - Intra-chain: Loop over inverse temperatures
    - propose regular MCMC move
    - propose or accept
  - Periodically (`nFreqSwap`) swap chains
    - propose swap between chains i and j
    - swap temperatures if move accepted
    - if swap accepted, update chain information
  - Periodically (`thin`) save state of the chains

In addition, we'd like to store summaries of the acceptance ratio for swapping and intra-chain moves.

```python
def mcmcmc(x0, a, gamma = 16., delta = 0.25, nsteps = 1000,
           thin = 10, nFreqSwap = 10):
    """Main MCMC routine: x0 = initial state; a =
    inital invTemp vector; delta = step size for
    ordinary Metropolis; nsteps = #MCS"""

    # initialize
    nchain   = len(a)
    z        = np.ones((nchain,1)) * x0
    hz       = pU(z, gamma)

    recz     = np.zeros(int(nsteps/thin, nchain))  # record
    recz[0]  = z.reshape((1,nchain))

    # number of successes
    nSucInt  = np.zeros((nchain))  # inter-chain
    nSucExc  = np.zeros((nchain-1)) # swap
```

```
# regular MCS cycle
for iMCS in range(1,nsteps):

    # cycle through temperatures
    for iTemp in range(nchain):

        x = z[iTemp]    # present values of chain
        h = hz[iTemp]

        # regular intra-chain Metropolis
        newx = metro_proposal(x, delta)
        accept, newh = metro_accept(newx, h, pU, gamma,
                                    a[iTemp])

        if accept:
            z[iTemp]  = newx
            hz[iTemp] = newh
            nSucInt[iTemp] += 1
    # swapping
    if iMCS  % nFreqSwap == 0:
        i, j = proposeSwap(nchain) # propose
        if i != j:
            accept = swap_accept(hz[i], hz[j], a[i], a[j])

            if accept:   # book-keeping
                nSucExc[min(i,j)] += 1
                tmp = z[i];  tmph  = hz[i]
                z[i] = z[j]; hz[i] = hz[j]
                z[j] = tmp;  hz[j] = tmph

    if iMCS % thin == 0:         # printing
        recz[int(iMCS/thin),:] = z.reshape((1,nchain))

return recz, nSucInt, nSucExc
```

Results

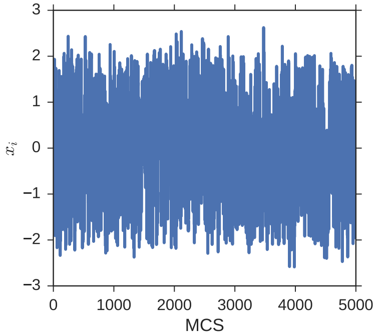
Initially I set `nFreqSwap = nsteps = 50000` so that no mixing moves were proposed

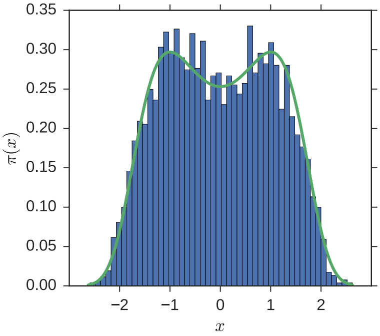
```
a = np.array([1.0, 0.5, 0.2, 0.01])
gam = 16.0

# case without swapping; 4 chains in parallel
recz, ar_int, ar_exc=mcmcmc(0., a, nsteps=50000,
                            nFreqSwap = 50000)

Intra-chain # success
[ 20052.  25666.  33960.  46709.]
# Success for Exchange involving chain
[ 0.  0.  0.]
```

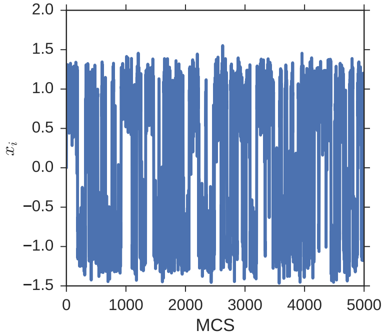
Hot Chain $a = 0.01$

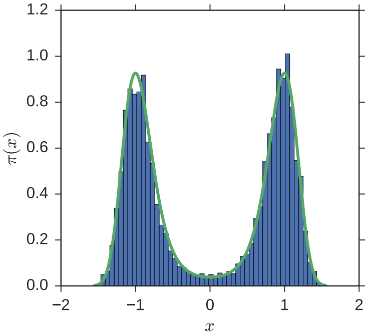




Good and rapid mixing

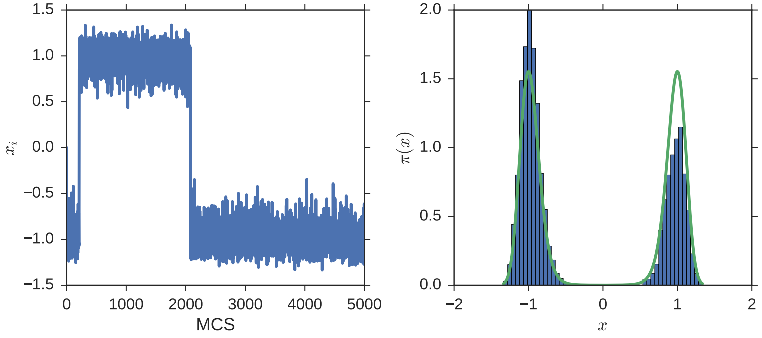
Warm Chain $a = 0.2$





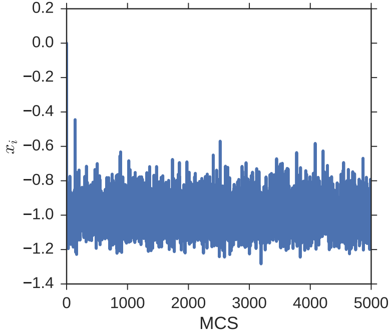
Good and rapid mixing

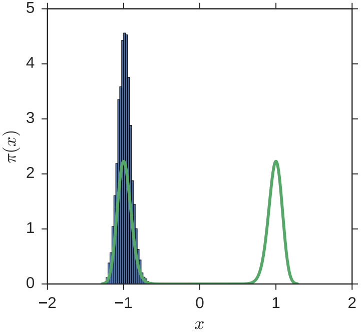
Cool Chain $a = 0.5$



Bad mixing

Cold Chain $a = 1.0$





Disaster!

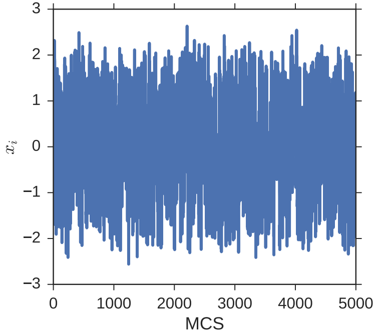
Mixing On

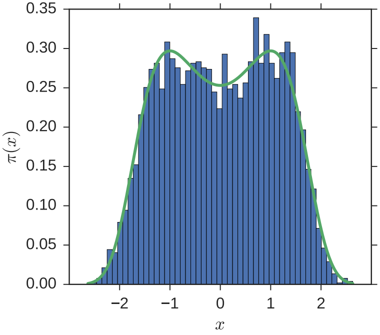
Now let's repeat the same with mixing every 10 MCS

```
# case with swapping
recz, ar_int, ar_exc=mcmcmc(0., a, nsteps=50000, thin=10,
                                nFreqSwap = 10)
```

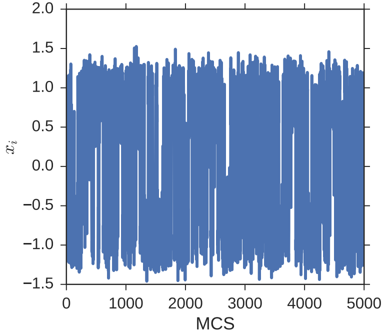
```
Intra-chain # successes
[ 19856.  25482.  33819.  46634.]
# Success for Exchange involving chain
[ 943.  876.  529.]
```

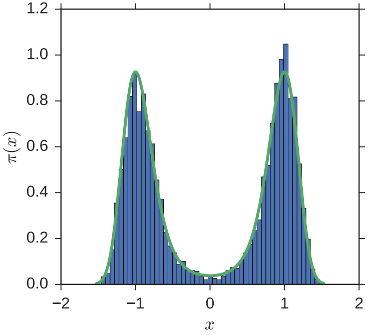
Hot Chain $a = 0.01$



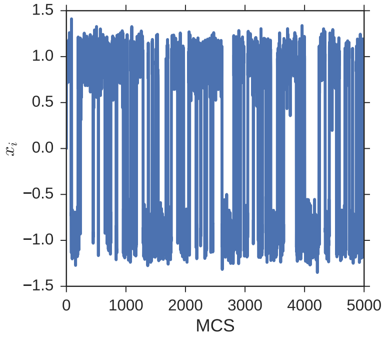


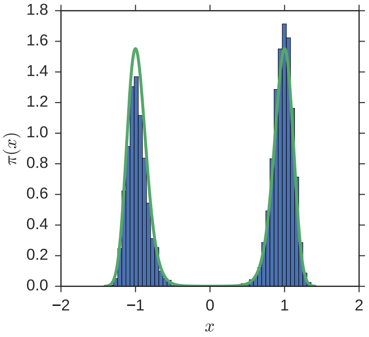
Warm Chain $a = 0.2$



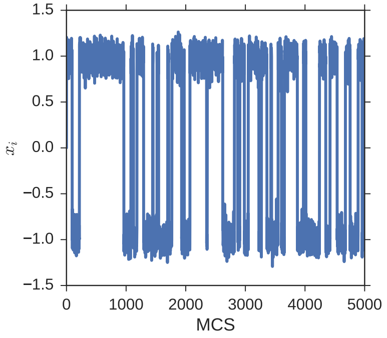


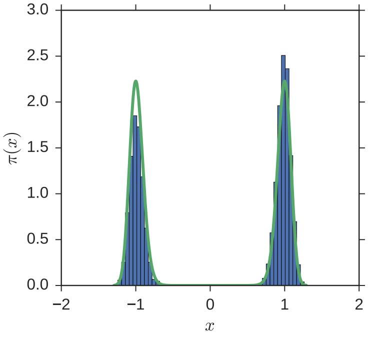
Cool Chain $a = 0.5$





Cold Chain $a = 1.0$





Both peaks explored!

Summary

- MC3 is a potential tool for improving chain mixing
- Particularly useful for multi-modal distributions, which tend to be the hardest sampling problems.
- Can be easily parallelized on multicore computers
- A geometric distribution of the inverse temperatures is generally a good first guess. They can be further adjusted to get desired acceptance rates for chain exchange moves.
- The intra-chain proposal steps for chains at different temperatures can be different; they may be adjusted to have comparable acceptance ratios.