Markov Chain Monte Carlo

The Metropolis Algorithm

Sachin Shanbhag

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

Motivating Example

Consider sampling the simple 1D distribution

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

We could use direct sampling to sample this distribution.

Direct Sampling

(i) Transformation Method: If $u \sim U[0,1]$, then we need to solve

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

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

(ii) Accept-Reject Method
Alternatively, enclose the PDF within a box of height height $\approx$ 2.0, and throw darts at it.

For such simple $\pi(x)$ direct sampling methods work well, and should indeed be preferred.

MCMC

We will consider how to apply an MCMC method to sample this distribution.

We expect this method to be inferior to direct sampling for simple problems.

Later, we will 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 teach how MCMC works.

In some ways, this is akin to MC versus quadrature methods. For simple low-dimensional problems, quadrature methods work better.

For simple low-dimensional sampling, direct MC algorithms can be constructed and should be preferred.

Metropolis MCMC

Consider a point $x_c \in [0, 1]$ in the domain of $\pi(x)$.

We propose a nearby point randomly using,

$$x_n = x_c + \delta \times U[-1, 1].$$

Formally, this can be represented as:

$$\pi(x_n|x_c) = U[x_c - \delta, x_c + \delta].$$

Metropolis MCMC: Detailed Balance

We can't simply accept x_n; somehow we need to bring in the distribution we seek to sample $\pi(x)$.

Similar to rejection sampling in direct MC, proposed moves in MCMC, are accepted or rejected based on some criterion that ensures we sample $\pi(x)$.

In MCMC, detailed balance provides the necessary guidance.

Recall detailed balance requires "net traffic" between any two states to be zero.

For the two points x_c and x_n, this implies:

$$W(x_c \to x_n)\pi(x_c) = W(x_n \to x_c)\pi(x_n). \tag{1}$$

Detailed Balance

The net transition probabilities can be decomposed into the proposal p and acceptance a contributions:

$$W(x_c \to x_n) = p(x_c \to x_n)a(x_c \to x_n). \tag{2}$$

In Metropolis MCMC, the proposal functions are symmetric:

$$\begin{aligned} p(x_c \to x_n) &= U[x_c - \delta, x_c + \delta] = \frac{1}{2\delta} \\ p(x_n \to x_c) &= U[x_n - \delta, x_n + \delta] = \frac{1}{2\delta} \end{aligned} \tag{3}$$

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

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

Metropolis MCMC

Thus, equation 1 can be simplified using 2:

$$a(x_c \to x_n)\pi(x_c) = a(x_n \to x_c)\pi(x_n), \tag{4}$$

$$\frac{a(x_c \to x_n)}{a(x_n \to x_c)} = \frac{\pi(x_n)}{\pi(x_c)}. \tag{5}$$

Metropolis et al. suggested that *one way of satisfying* detailed balance (eqn. 5) was to choose:

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

Or in shorthand:

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

Why does this work?

To see why this choice works, consider the two cases

(i) $\pi(x_n) \geq \pi(x_c)$. In this case,

$$
\begin{aligned}
a(x_c \to x_n) &= 1, \text{ and} \\
a(x_n \to x_c) &= \pi(x_c)/\pi(x_n). \\
\implies \frac{a(x_c \to x_n)}{a(x_n \to x_c)} &= \frac{1}{\pi(x_c)/\pi(x_n)} = \frac{\pi(x_n)}{\pi(x_c)},
\end{aligned}
$$

as required by detailed balance (eqn. 5)

(ii) $\pi(x_n) < \pi(x_c)$. Here,

$$
\begin{aligned}
a(x_c \to x_n) &= \pi(x_n)/\pi(x_c), \text{ and} \\
a(x_n \to x_c) &= 1. \\
\implies \frac{a(x_c \to x_n)}{a(x_n \to x_c)} &= \frac{\pi(x_n)}{\pi(x_c)}
\end{aligned}
$$

Warning on Notation

Note: Here I expanded the transition from state x_c to state x_n as $x_c \to x_n$, so that there is no notational confusion.

Remember that people often write $p(x_c \to x_n)$ as $p(x_n|x_c)$, or even p_{nc}.

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

$$W(j \to i) = W_{ij}.$$

Technical Note

Q: What TPM does Metropolis MCMC lead to?

This is a theoretical concern, since practically we don't have to deal with the TPM itself.

In Metropolis, 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 $i = j$.

This rejection implies that $W_{ij} = p_{ij} a_{ij}$ only for "off-diagonal" terms ($j \neq i$).

The diagonal term can be non-zero even if "self-moves" are not explicity proposed ("$p_{jj} = 0$"); it accumulates weight due to rejection of proposed moves.

Technical Note

Formally,

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

where the second term captures the effect of rejections.

δ_{ij} is the Kronecker delta function; it does not contribute to off-diagonal terms.

The summation extends over all possible states i' that are accessible from state j.

Note that the TPM depends on the choice of proposal and acceptance functions p_{ij} and a_{ij}.

We don't need the normalization factor of the target distribution $\pi(x)$, since we only deal with ratios.

Big Picture

- Balance is mandatory for MCMC to work
- Detailed balance is a sufficient condition that is almost always imposed.
- Metropolis MCMC is a particular choice/algorithm for implementing detailed balance.

Big Picture

There are other choices, including, for example, Glauber "dynamics"

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

Exercise: Show that this choice also satisfies eqn. 5

Exercise: If $\pi_n/\pi_c = 1$, what is $a(c \to n)$ according to Metropolis and Glauber.
Answer: Metropolis $a(c \to n) = 1$, Glauber $a(c \to n) = 0.5$

Exercise: Plot $a(c \to n)$ as a function of π_n/π_c for Metropolis and Glauber.

Code

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

1. Probability Distribution

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

Code

2. Proposal Function:

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

3. Acceptance function

```
def 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
```

Code: Driver

```
def metropolisMCMC(nmcs, delta, x0):

    x     = np.zeros((nmcs)) # samples
    xc    = x0               # initial state
    nSucc = 0                # #accept

    for imcs in range(nmcs):

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

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

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

Results

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

acceptance ratio $= 0.83$

Results

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

acceptance ratio $= 0.8264$

Issues

Metropolis MCMC is extremely powerful, versatile, and ubiquitous.

For many, relatively benign problems (like this example), this is all the MCMC you need to know!

Even so, we have to deal with a multitude of issues:

- need to specify proposal $p_{nc} = \pi(x_n|x_c)$
  - uniform, normal, etc.
  - symmetric or asymmetric (Metropolis-Hastings)
- need to specify δ
  - too small - Markov chain doesn't travel too far; choice may be too conservative
  - too large - acceptance ratio falls; choice may be too aggressive

Issues

- the sequence of x_i generated are not independent
  - correlation
  - complicates error analysis
  - one approach: block averaging
- there is often a burn-in period
  - the first few samples are tainted by choice of x_0
  - start analyzing samples after Markov chain is stationary
- has my MCMC simulation converged?
  - governed by the magnitude of the second largest eigenvalue of $\mathbf{W}$
  - in practice this is a hard question with many answers; but none completely bullet-proof

Issues

Hopefully, by the time we finish the course, you will come to understand and respect some of these issues.

In addition we will keep adding to our suite of algorithms for MCMC.

A good MCMC algorithm:

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

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

Next let us consider a "three-peaks" problem, which allows us to explore some of these issues (credit: Peter Beerli)

Example: Three Peaks

Consider the following 2D PDF with three peaks:

$$\pi(x,y) \sim \exp\left(-\frac{1}{10}(x^2+y^2)\right) + \frac{1}{2}\exp\left(-\frac{1}{20}((x-20.)^2+(y-20.)^2)\right) + \exp\left(-\frac{1}{200}((x-40.)^2+(y-40.)^2)\right) \tag{6}$$

```
def fxy(x,y):
    """definition of function"""
    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))
    return f
```

Plot

Metropolis MCMC

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.

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

We already presented the code for the function ($\pi(x, y)$); now let us write the

- proposal
- acceptance, and
- driver functions.

Proposal Function

We entertain a domain $\Omega = [-30, 100]^2$ (driver)

$$(x', y') = (U[x - \delta, x + \delta], U[y - \delta, y + \delta])$$

```
def proposal(x, y, delta, domain):
    """proposal function"""

    newx = x + delta * (2. * np.random.rand() - 1.)
    newy = y + delta * (2. * np.random.rand() - 1.)

    # if point inside domain, propose
    if (domain[0][0] < newx < domain[0][1]) and
       (domain[1][0] < newy < domain[1][1]):
        return newx, newy
    else: # else try again, recursion
        proposal(x, y, delta, domain)
```

Acceptance Function

standard Metropolis criterion

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

    numer = fxy(newx, newy);
    denom = fxy(oldx, oldy);

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

    return accept, x, y
```

Driver...

```
# preliminaries
domain = np.array([[-30.,100.],[-30.,100.]])
nsteps = 10000 # number of steps in the mcmc chain
delta  = 10.   # delta for the proposal
thin   = 10    # save every 10th point

# initial state
x = 0.
y = 0.

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

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

Driver (contd)

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

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

AccRatio = float(NumSucc)/float(nsteps)
print(AccRatio, NumSucc)
```

Results

Plot the results; overlay with $\pi(x, y)$ to be sampled.

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

x = np.linspace(-10,70,100)
X, Y = np.meshgrid(x,x)
Z = fxy(X,Y)
Z = Z.reshape(X.shape)

plt.contourf(X,Y,Z, cmap=cm.OrRd)
```

Results