A Simple Gibbs Sampler

Biostatistics 615/815
Lecture 20
Scheduling

- No Class
  - November 30

- Review session
  - Thursday, December 7

- Final Assessment
  - Tuesday, December 12
Optimization Strategies

- Single Variable
  - Golden Search
  - Quadratic Approximations

- Multiple Variables
  - Simplex Method
  - E-M Algorithm
  - Simulated Annealing
Simulated Annealing

- Stochastic Method

- Sometimes takes up-hill steps
  - Avoids local minima

- Solution is gradually *frozen*
  - Values of parameters with largest impact on function values are fixed earlier
Gibbs Sampler

- Another MCMC Method
- Update a single parameter at a time
- Sample from conditional distribution when other parameters are fixed
Gibbs Sampler Algorithm

Consider a particular choice of parameter values $\theta^{(t)}$

Define the next set of parameter values by:

a. Selecting component to update, say $i$

b. Sample value for $\theta_i^{(t+1)}$ from $p(\theta_i \mid x, \theta_1, \theta_2, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_k)$

Increment $t$ and repeat previous steps.
**Alternative Algorithm**

Consider a particular choice of parameter values $\theta^{(t)}$

Define the next set of parameter values by:

a. Update each component, $1 \ldots k$, in turn
b. Sample value for $\theta_1^{(t+1)}$ from $p(\theta_1 | x, \theta_2, \theta_3, \ldots \theta_k)$
c. Sample value for $\theta_2^{(t+1)}$ from $p(\theta_2 | x, \theta_1, \theta_3, \ldots \theta_k)$
   ...
   z. Sample value for $\theta_k^{(t+1)}$ from $p(\theta_k | x, \theta_1, \theta_3, \ldots \theta_{k-1})$

Increment $t$ and repeat previous steps.
Key Property: Stationary Distribution

Suppose that \((\theta_1^{(i)}, \theta_2^{(i)}, \ldots, \theta_k^{(i)}) \sim p(\theta_1, \theta_2, \ldots, \theta_k | x)\)

Then \((\theta_1^{(i+1)}, \theta_2^{(i)}, \ldots, \theta_k^{(i)})\) is distributed as

\[
p(\theta_1 | \theta_2, \ldots, \theta_k, x) p(\theta_2, \ldots, \theta_k | x) = p(\theta_1, \theta_2, \ldots, \theta_k | x)
\]

In fact...

\[
\theta^{(i)} \sim p(\theta | x) \Rightarrow \theta^{(i+1)} \sim p(\theta | x)
\]

Eventually, we expect the Gibbs sampler to sample parameter values from their posterior distribution.
Gibbs Sampling for Mixture Distributions

- Sample each of the mixture parameters from conditional distribution
  - Dirichlet, Normal and Gamma distributions are typical

- Simple alternative is to sample the source of each observation
  - Assign observation to specific component
Sampling A Component

\[
\Pr(Z_j = i \mid x_j, \pi, \varphi, \eta) = \frac{\pi_i f(x_j \mid \phi_i, \eta)}{\sum_l \pi_l f(x_j \mid \phi_l, \eta)}
\]

- Calculate the probability that the observation originated from a specific component…

- … can you recall how random numbers can be used to sample from one of a few discrete categories?
C Code: Sampling A Component

```c
int sample_group(double x, int k,
    double * probs, double * mean, double * sigma)
{
    int group; double p = Random();
    double lk = dmix(x, k, probs, mean, sigma);

    for (group = 0; group < k - 1; group++)
    {
        double pgroup = probs[group] *
            dnorm(x, mean[group], sigma[group])/lk;

        if (p < pgroup)
            return group;

        p -= pgroup;
    }

    return k - 1;
}
```
Calculating Mixture Parameters

\[ n_i = \sum_{j: Z_j = i} 1 \]
\[ p_i = \frac{n_i}{n} \]
\[ \bar{x}_i = \frac{\sum_{j: Z_j = i} x_j}{n_i} \]
\[ s_i = \sqrt{\left( \sum_{j: Z_j = i} x_j^2 - n_i \bar{x}_i^2 \right) / n_i} \]

- Before sampling a new origin for an observation…
- … update mixture parameters given current assignments
- Could be expensive!
### C Code: Updating Parameters

```c
void update_estimates(int k, int n,
                        double * prob, double * mean, double * sigma,
                        double * counts, double * sum, double * sumsq)
{
    int i;

    for (i = 0; i < k; i++)
    {
        prob[i] = counts[i] / n;
        mean[i] = sum[i] / counts[i];
        sigma[i] = sqrt((sumsq[i] - mean[i]*mean[i]*counts[i])
                        / counts[i] + 1e-7);
    }
}
```
```c
void remove_observation(double x, int group,
                        double * counts, double * sum, double * sumsq)
{
    counts[group] --;
    sum[group] -= x;
    sumsq[group] -= x * x;
}

void add_observation(double x, int group,
                      double * counts, double * sum, double * sumsq)
{
    counts[group] ++;
    sum[group] += x;
    sumsq[group] += x * x;
}
```
Selecting a Starting State

- Must start with an assignment of observations to groupings
- Many alternatives are possible, I chose to perform random assignments with equal probabilities...
void initial_state(int k, int * group, double * counts, double * sum, double * sumsq)
{
    int i;

    for (i = 0; i < k; i++)
        counts[i] = sum[i] = sumsq[i] = 0.0;

    for (i = 0; i < n; i++)
    {
        group[i] = Random() * k;

        counts[group[i]] ++;
        sum[group[i]] += data[i];
        sumsq[group[i]] += data[i] * data[i];
    }
}
The Gibbs Sampler

- Select initial state
- Repeat a large number of times:
  - Select an element
  - Update conditional on other elements
- If appropriate, output summary for each run…
initial_state(k, probs, mean, sigma, group, counts, sum, sumsq);
for (i = 0; i < 10000000; i++)
{
    int id = Random() * n;

    if (counts[group[id]] < MIN_GROUP) continue;

    remove_observation(data[id], group[id], counts, sum, sumsq);
    update_estimates(k, n - 1, probs, mean, sigma,
                      counts, sum, sumsq);
    group[id] = sample_group(data[id], k, probs, mean, sigma);
    add_observation(data[id], group[id], counts, sum, sumsq);

    if ((i > BURN_IN) && (i % THIN_INTERVAL == 0))
        /* Collect statistics */
}
Gibbs Sampler: Memory Allocation and Freeing

```c
void gibbs(int k, double * probs, double * mean, double * sigma)
{
    int i, * group = (int *) malloc(sizeof(int) * n);
    double * sum = alloc_vector(k);
    double * sumsq = alloc_vector(k);
    double * counts = alloc_vector(k);

    /* Core of the Gibbs Sampler goes here */

    free_vector(sum, k);
    free_vector(sumsq, k);
    free_vector(counts, k);
    free(group);
}
```
Example Application
Old Faithful Eruptions (n = 272)
My first few runs found excellent solutions by fitting components accounting for very few observations but with variance near 0.

Why?
- Repeated values due to rounding
- To avoid this, I set MIN_GROUP to 13 (5%)
Gibbs Sampler Burn-In

LogLikelihood

-450
-400
-350
-300
-250
0 2500 5000 7500 10000

Iteration

Log-Likelihood

-450
-400
-350
-300
-250

0 2500 5000 7500 10000
Gibbs Sampler Burn-In

Mixture Means

Iteration

Mean

0 2500 5000 7500 10000
Gibbs Sampler After Burn-In Likelihood
Gibbs Sampler After Burn-In
Mean for First Component

![Graph showing iteration vs component mean](image-url)
Gibbs Sampler After Burn-In
Notes on Gibbs Sampler

- Previous optimizers settled on a minimum eventually.

- The Gibbs sampler continues wandering through the stationary distribution…

- Forever!
Drawing Inferences...

- To draw inferences, summarize parameter values from stationary distribution.

- For example, might calculate the mean, median, etc.
Component Means
Component Probabilities
Overall Parameter Estimates

The means of the posterior distributions for the three components were:
- Frequencies of 0.073, 0.278 and 0.648
- Means of 1.85, 2.08 and 4.28
- Variances of 0.001, 0.065 and 0.182

Our previous estimates were:
- Components contributing 0.160, 0.195 and 0.644
- Component means are 1.856, 2.182 and 4.289
- Variances are 0.00766, 0.0709 and 0.172
Joint Distributions

- Gibbs Sampler provides other interesting information and insights

- For example, we can evaluate joint distribution of two parameters...
Component Probabilities

![Graph showing the relationship between Group 1 Probability and Group 3 Probability]

- X-axis: Group 1 Probability
- Y-axis: Group 3 Probability
So far today ...

- Introduction to Gibbs sampling
- Generating posterior distributions of parameters conditional on data
- Providing insight into joint distributions
A little bit of theory

- Highlight connection between Simulated Annealing and the Gibbs sampler ...

- Fill in some details of the Metropolis algorithm
Both Methods Are Markov Chains

- The probability of any state being chosen depends only on the previous state

\[
\Pr(S_n = i_n \mid S_{n-1} = i_{n-1}, \ldots, S_0 = i_0) = \Pr(S_n = i_n \mid S_{n-1} = i_{n-1})
\]

- States are updated according to transition matrix with elements \( p_{ij} \). This matrix defines important properties, including \textit{periodicity} and \textit{irreducibility}. 
Metropolis-Hastings Acceptance Probability

Let $q_{ij} = q(\text{propose } S_{n+1} = j \mid S_n = i)$

Let $\pi_i$ and $\pi_j$ be the relative probabilities of each state

The Metropolis-Hastings acceptance probability is:

$$a_{ij} = \min \left( 1, \frac{\pi_j q_{ji}}{\pi_i q_{ij}} \right) \text{ or } a_{ij} = \min \left( 1, \frac{\pi_j}{\pi_i} \right) \text{ if } q_{ij} = q_{ji}$$

Only the ratio $\frac{\pi_j}{\pi_i}$ must be known, not the actual values of $\pi$
Metropolis-Hastings Equilibrium

If we use the Metropolis-Hastings algorithm to update a Markov Chain, it will reach an equilibrium distribution where \( \Pr(S = i) = \pi_i \)

For this to happen, the proposal density must allow all states to communicate.
The Gibbs sampler ensures that $\pi_i q_{ij} = \pi_j q_{ji}$.

As a consequence, $a_{ij} = \min \left( 1, \frac{\pi_j q_{ji}}{\pi_i q_{ij}} \right) = 1$.
Simulated Annealing

Given a temperature parameter $\tau$,

replace $\pi_i$ with $\pi_i^{(\tau)} = \frac{\frac{1}{\pi_i^\tau}}{\sum_j \pi_j^\tau}$

At high temperatures, the probability distribution is flattened
At low temperatures, larger weights are given to high probability states
Additional Reading

- If you need a refresher on Gibbs sampling
  - Bayesian Methods for Mixture Distributions
    M. Stephens (1997)
    http://www.stat.washington.edu/stephens/

- Numerical Analysis for Statisticians
  - Kenneth Lange (1999)
  - Chapter 24