Simulated Annealing

Biostatistics 615/815

Lecture 19

Scheduling

- Need to pick a date for mid-term
- Default date is December 20, 2006
- We could have it earlier…
  - For example, on December 12, 2006?
- What do you prefer?

So far …

- "Greedy" optimization methods
  - Can get trapped at local minima
  - Outcome might depend on starting point

- Examples:
  - Golden Search
  - Nelder-Mead Simplex Optimization
  - E-M Algorithm

Today …

- Simulated Annealing
- Markov-Chain Monte-Carlo method
- Designed to search for global minimum among many local minima

The Problem

- Most minimization strategies find the *nearest* local minimum

- Standard strategy
  - Generate trial point based on current estimates
  - Evaluate function at proposed location
  - Accept new value if it improves solution

The Solution

- We need a strategy to find other minima
- This means, we must sometimes select new points that do not improve solution
- How?

Annealing

- One manner in which crystals are formed
- Gradual cooling of liquid …
  - At high temperatures, molecules move freely
  - At low temperatures, molecules are "stuck"
- If cooling is slow
  - Low energy, organized crystal lattice formed

Simulated Annealing

- Analogy with thermodynamics
- Incorporate a temperature parameter into the minimization procedure
- At high temperatures, explore parameter space
- At lower temperatures, restrict exploration

Markov Chain

- Start with some sample
  - A set of mixture parameters
- Propose a change
  - Edit mixture parameters in some way
- Decide whether to accept change
  - Decision is based on relative probabilities of two outcomes

Simulated Annealing Strategy

- Consider decreasing series of temperatures
- For each temperature, iterate these steps:
  - Propose an update and evaluate function
  - Accept updates that improve solution
  - Accept some updates that don't improve solution
    - Acceptance probability depends on "*temperature*" parameter
- If cooling is sufficiently slow, the global minimum will be reached

Example Application

- The traveling salesman problem
  - Salesman must visit every city in a set
  - Given distances between pairs of cities
  - Find the shortest route through the set
- No practical deterministic algorithms for finding optimal solution are known…
  - … simulated annealing and other stochastic methods can do quite well

Update Scheme

- A good scheme should be able to:
  - Connect any two possible paths
  - Propose improvements to good solutions
- Some possible update schemes:
  - Swap a pair of cities in current path
  - Invert a segment in current path
- What do you think of these?

How simulated annealing proceeds ...

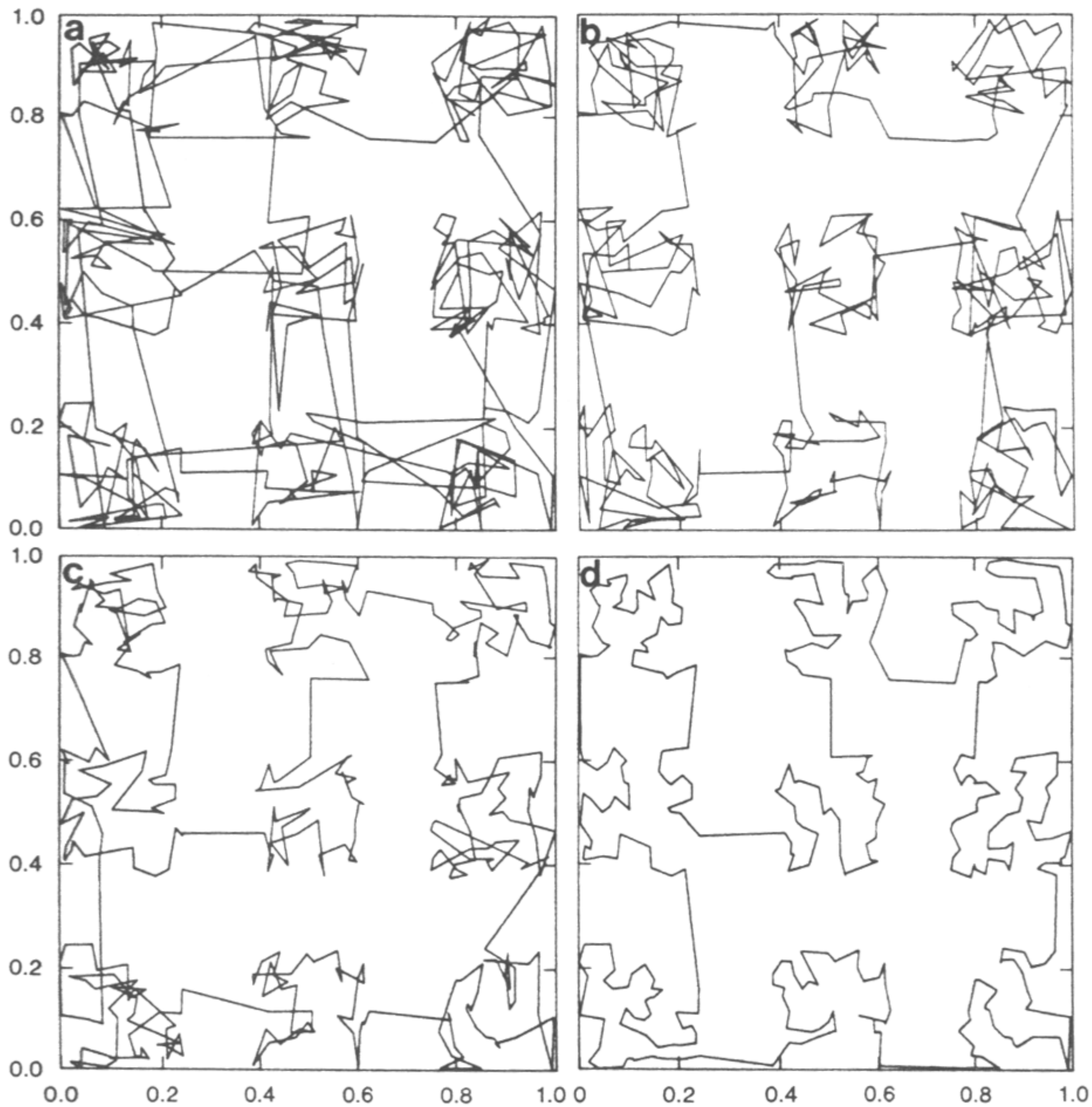


Fig. 9. Results at four temperatures for a clustered 400-city traveling salesman problem. The points are uniformly distributed in nine regions. (a) $T = 1.2$, $\alpha = 2.0567$; (b) $T = 0.8$, $\alpha = 1.515$; (c) $T = 0.4$, $\alpha = 1.055$; (d) $T = 0.0$, $\alpha = 0.7839$.

A little more detail

- Metropolis (1953), Hastings (1970)
  - Define a set of conditions that, if met, ensure the random walk will sample from probability distribution at equilibrium
    - In theory
  - Recommendations apply to how changes are proposed and accepted

Accepting an Update

- The *Metropolis criterion*
- Change from E_0 to E with probability

$$\min\left(1, \exp\left\{-\frac{(E-E_0)}{T}\right\}\right)$$

- Given sufficient time, leads to equilibrium state

Evaluating Proposals in Simulated Annealing

```
int accept_proposal(double current, double proposal,
                    double temperature)
   {
   double prob;

   if (proposal < current)
      return 1;

   if (temperature == 0.0)
      return 0;

   prob = exp(-(proposal - current) / temperature);

   return rand_prob() < prob;
   }
```

Key Requirement: Irreducibility

- All states must communicate
  - Starting point should not affect results
- If Q is matrix of proposal probabilities
  - Either $Q_{ij} > 0$ for all possible states i and j
  - Some integer P exists where $(Q^P)_{ij} > 0$ for all *i,j*

Equilibrium Distribution

- Probability of state with energy *k* is

$$P(E = k) \propto \exp\left(-\frac{k}{T}\right)$$

- At low T, probability is concentrated in low energy states

Simulated Annealing Recipe

1. Select starting temperature and initial parameter values

2. Randomly select a new point in the neighborhood of the original

3. Compare the two points using the *Metropolis criterion*

Simulated Annealing Recipe

4. Repeat steps 2 and 3 until system reaches equilibrium state…
   - In practice, repeat the process N times for large N

5. Decrease temperature and repeat the above steps, stop when system reaches frozen state

Practical Issues

- The maximum temperature
- Scheme for decreasing temperature
- Strategy for proposing updates

Selecting a Nearby Point

- Suggestion of Brooks and Morgan (1995) works well for our problem
  - Select a component to update
  - Sample from within plausible range

- Many other alternatives
  - The authors of *Numerical Recipes* use a variant of the Nelder-Mead method

C Code:
Simple Sampling Functions

```
// Assume that function Random() generates
// random numbers between 0.0 and 1.0
// Examples from lecture 14 are suitable

// Random numbers within arbitrary range
double randu(double min, double max)
    {
    return Random() * (max - min) + min;
    }
```

Updating Means and Variances

- Select component to update at random
- Sample a new mean (or variance) within plausible range for parameter
- Decide whether to accept proposal

C Code: Updating Means

```
double sa_means(int dim,
       double * probs, double * means, double * sigmas,
       double llk, double temperature, double min, double max)
   {
   int c = Random() * dim;
   double proposal, old = means[c];

   means[c] = randu(min, max);
   proposal = -mixLLK(n, data, dim, probs, means, sigmas);

   if (accept_proposal(llk, proposal, temperature))
      return proposal;

   means[c] = old;
   return llk;
   }
```

C Code:
Updating Standard Deviation

```
double sa_sigmas(int dim,
       double * probs, double * means, double * sigmas,
       double llk, double temperature, double min, double max)
   {
   int c = Random() * dim;
   double proposal, old = sigmas[c];

   sigmas[c] = randu(min, max);
   proposal = -mixLLK(n, data, dim, probs, means, sigmas);

   if (accept_proposal(llk, proposal, temperature))
      return proposal;

   sigmas[c] = old;
   return llk;
   }
```

Updating Mixture Proportions

- Mixture proportions must sum to 1.0
- When updating one proportion, must take others into account
- Select a component at random
  - Increase or decrease probability by ~20%
  - Rescale all proportions so they sum to 1.0

C Code:
Vector Utility Functions

```
double * duplicate_vector(double * v, int dim)
   {
   int i;
   double * dup = alloc_vector(dim);

   for (i = 0; i < dim; i++)
      dup[i] = v[i];

   return dup;
   }

void copy_vector(double * dest, double * source, int dim)
   {
   for (i = 0; i < dim; i++)
      dest[i] = source[i];
   }
```

C Code:
Changing Mixture Proportions

```
double sa_probs(int dim, double * probs, double * means,
                double * sigmas, double llk, double temperature)
   {
   int    i, c = Random() * dim;
   double proposal, * save_probs = duplicate_vector(probs, dim);

   probs[c] *= randu(0.8, 1.25);
   adjust_probs(probs, dim);

   proposal = -mixLLK(n, data, dim, probs, means, sigmas);
   if (accept_proposal(llk, proposal, temperature))
      llk = proposal;
   else
      copy_vector(probs, save_probs, dim);

   free_vector(save_probs, dim);
   return llk;
   }
```

C Code: Adjusting Probabilities

- The following function ensures probabilities always sum to 1.0

```
void adjust_probs(double * probs, int dim)
   {
   int i;
   double sum = 0.0;

   for (i = 0; i < dim; i++)
      sum += probs[i];

   for (i = 0; i < dim; i++)
      probs[i] /= sum;
   }
```

Simulated Annealing Procedure

- Cycle through temperatures

- At each temperature, evaluate proposed changes to mean, variance and mixing proportions

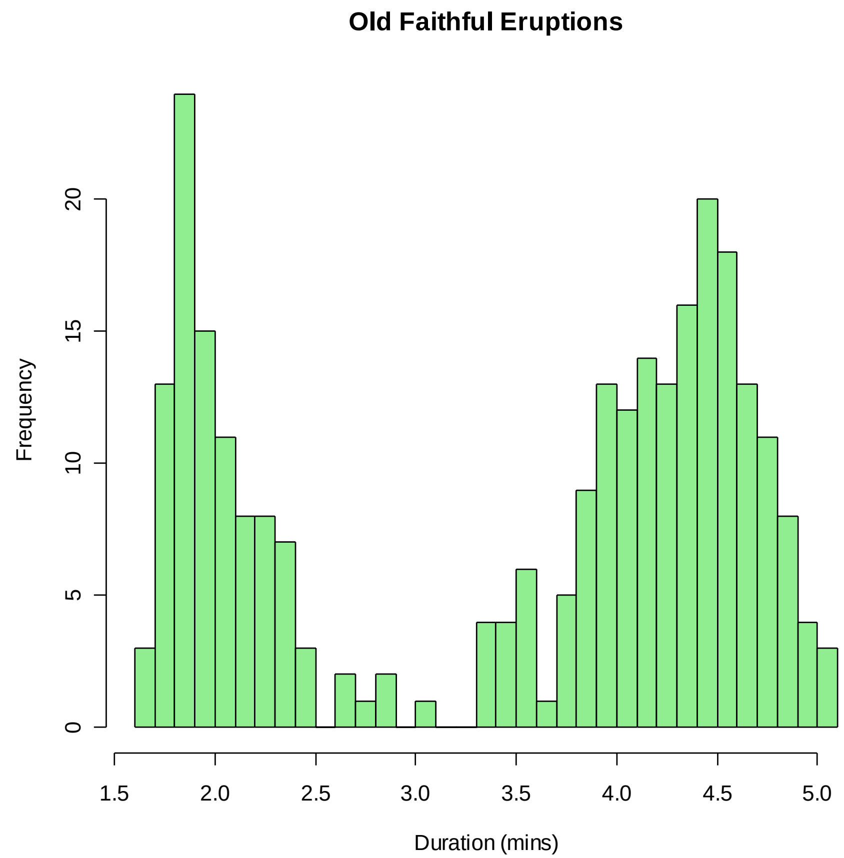
C Code: Simulated Annealing

```
double sa(int k, double * probs, double * means, double * sigmas, double eps)
   {
   double llk = -mixLLK(n, data, k, probs, means, sigmas);
   double temperature = MAX_TEMP; int choice, N;

   double lo = min(data, n), hi = max(data, n);
   double stdev = stdev(data, n), sdhi = 2.0 * stdev, sdlo = 0.1 * stdev;

   while (temperature > eps) {
      for (N = 0; N < 1000; N++)
         switch (choice = Random() * 3)
            {
            case 0 :
               llk = sa_probs(k, probs, means, sigmas, llk, temperature);
               break;
            case 1 :
               llk = sa_means(k, probs, means, sigmas, llk, temperature, lo, hi);
               break;
            case 2 :
               llk = sa_sigmas(k, probs, means, sigmas, llk, temperature, sdlo, sdhi);
            }
      temperature *= 0.90; }
   return llk;
   }
```

Example Application
Old Faithful Eruptions (n = 272)

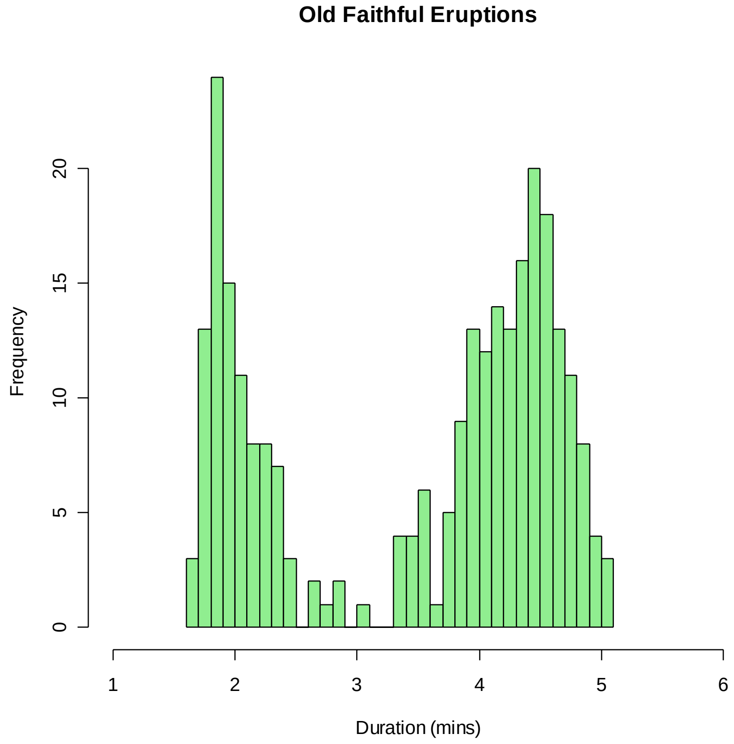


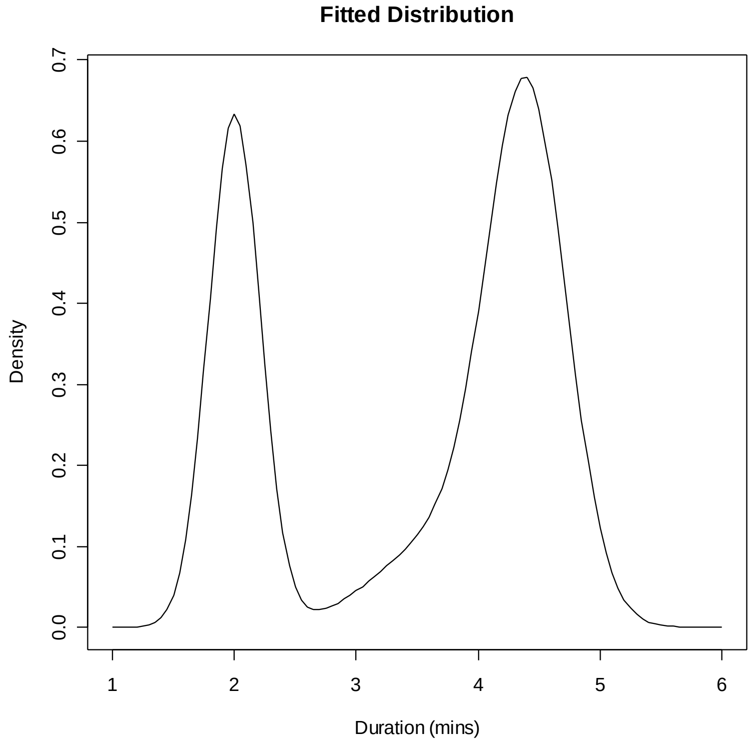
E-M Algorithm: A Mixture of Three Normals

- Fit 8 parameters
  - 2 proportions, 3 means, 3 variances
- Required about ~150 evaluations
  - Found log-likelihood of ~267.89 in 42/50 runs
  - Found log-likelihood of ~263.91 in 7/50 runs
- The best solutions …
  - Components contributing .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

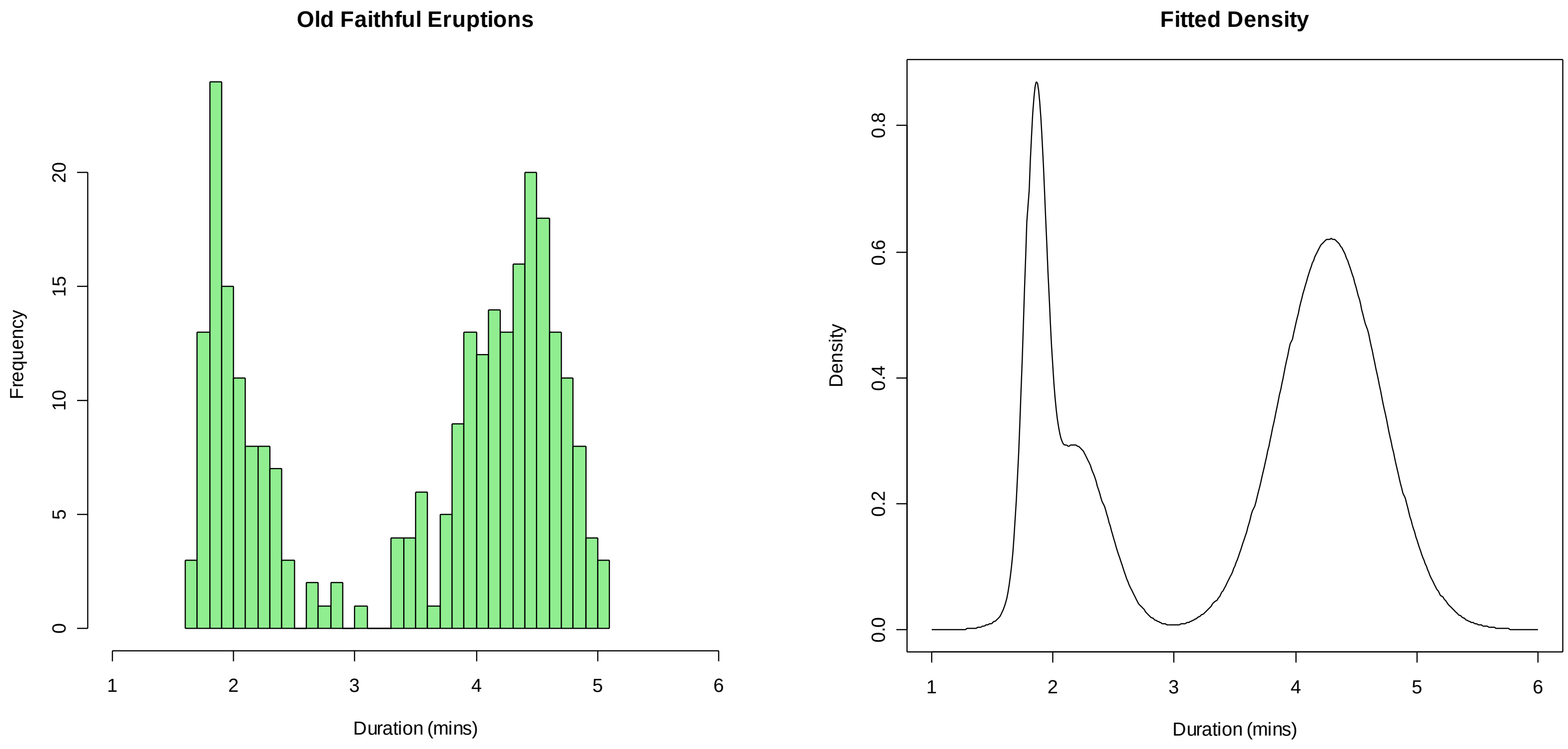
  - Maximum log-likelihood = -263.91

Three Components





Three Components



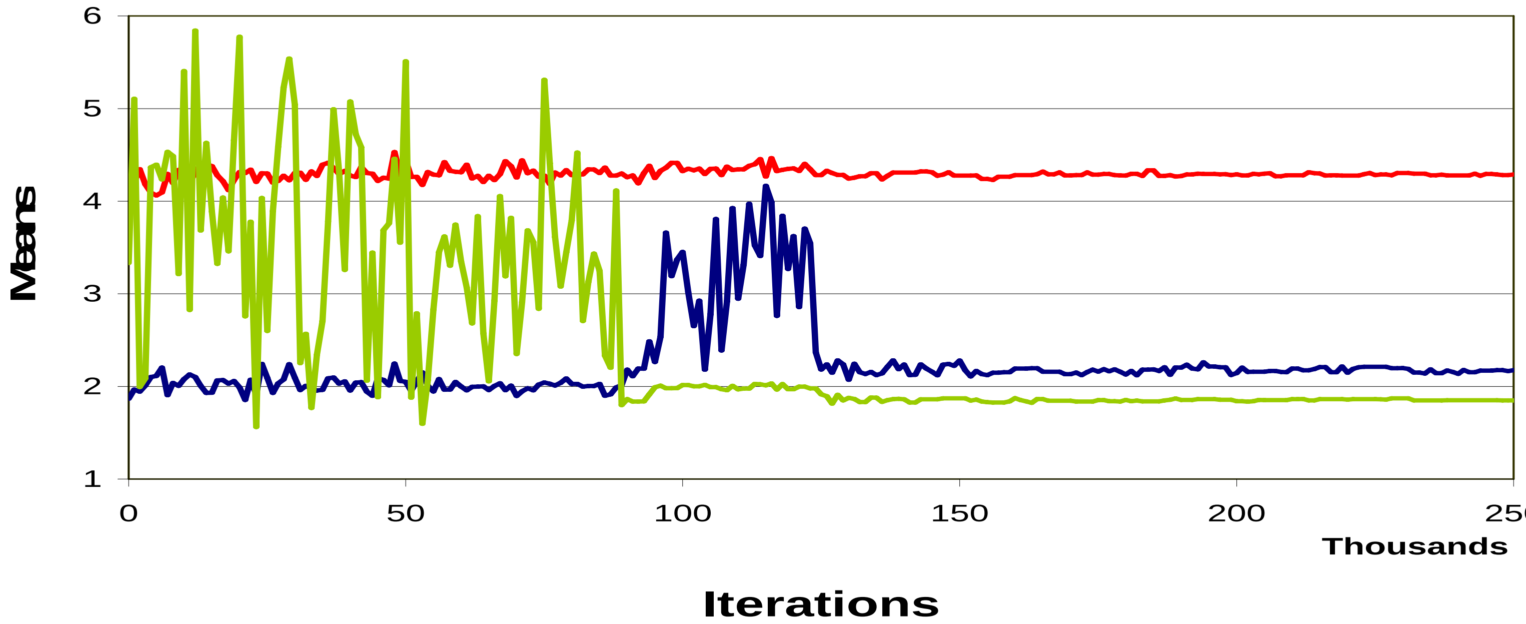
Simulated Annealing: Mixture of Three Normals

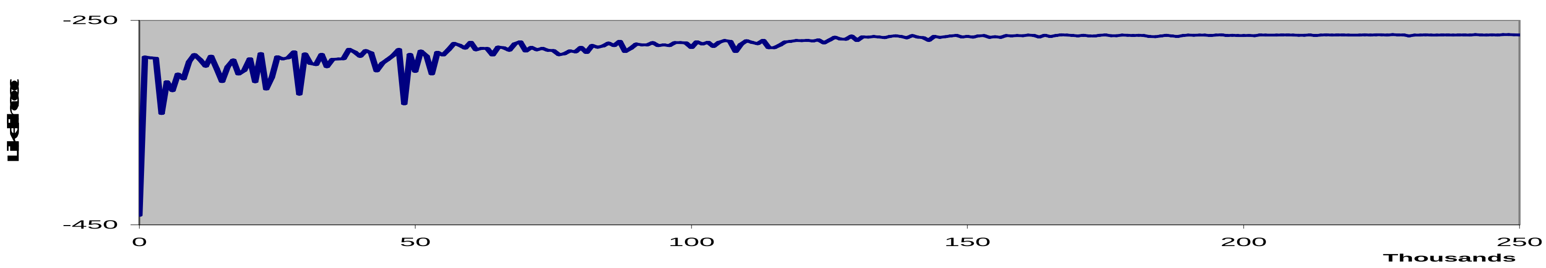
- Fit 8 parameters
  - 2 proportions, 3 means, 3 variances

- Required about ~100,000 evaluations
  - Found log-likelihood of ~267.89 in 30/50 runs
  - Found log-likelihood of ~263.91 in 20/50 runs
  - With slower cooling and 500,000 evaluations, minimum found in 32/50 cases

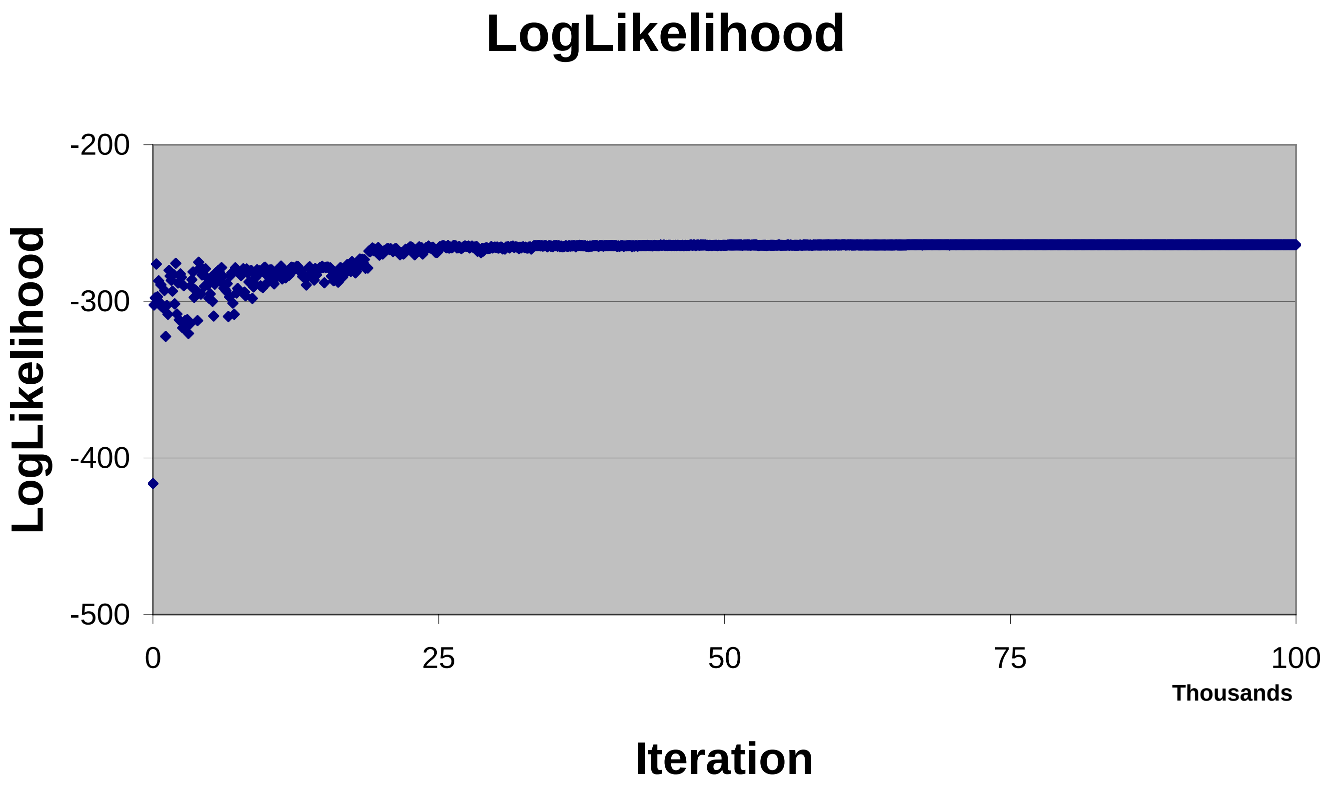
- 100,000 evaluations seems like a lot…
  - However, consider that even a 5 point grid search along 8 dimensions would require ~400,000 evaluations!

Convergence for Simulated Annealing

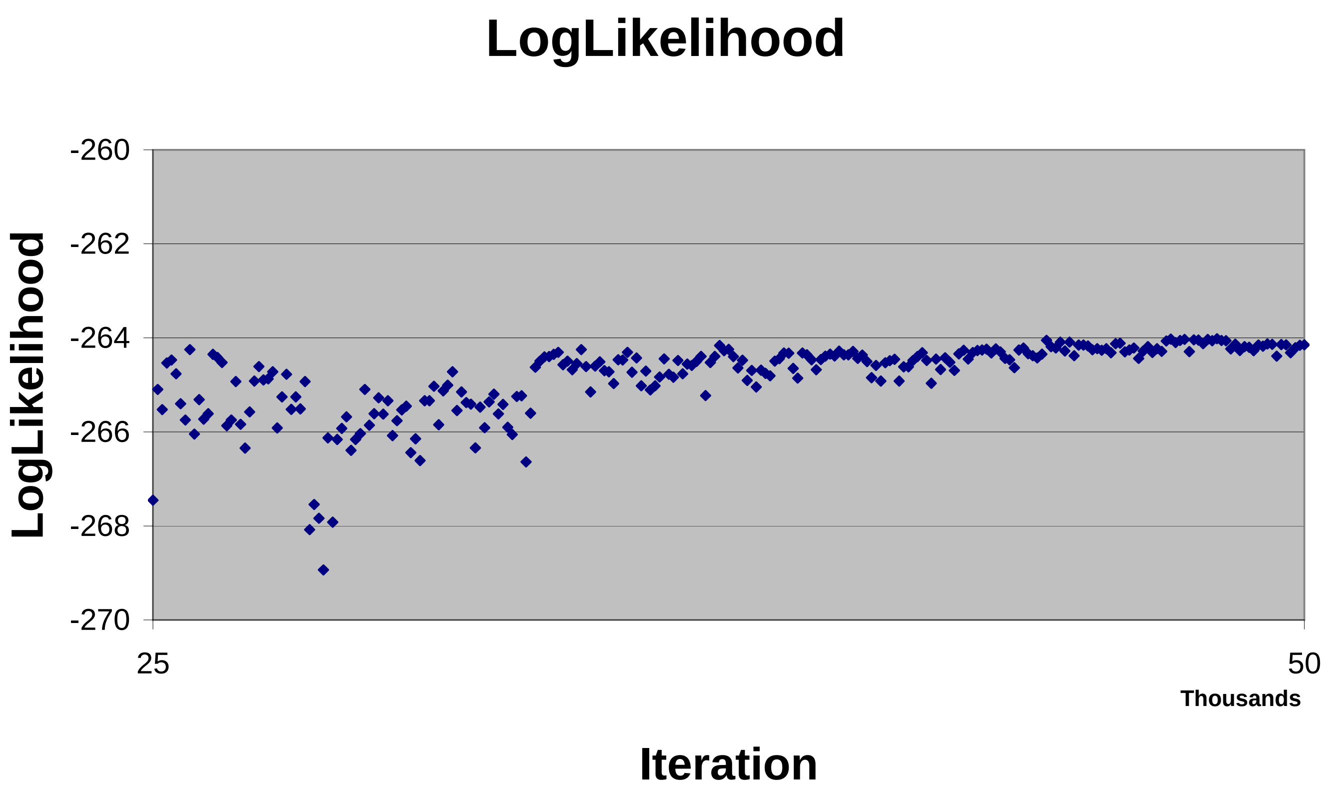




Convergence for Simulated Annealing



Convergence for Simulated Annealing



Importance of Annealing Step

- Evaluated a greedy algorithm
- Generated 100,000 updates using the same scheme as for simulated annealing
- However, changes leading to decreases in likelihood were never accepted
- Led to a minima in only 4/50 cases.

E-M Algorithm:
A Mixture of Four Normals

- Fit 11 parameters
  - 3 proportions, 4 means, 4 variances

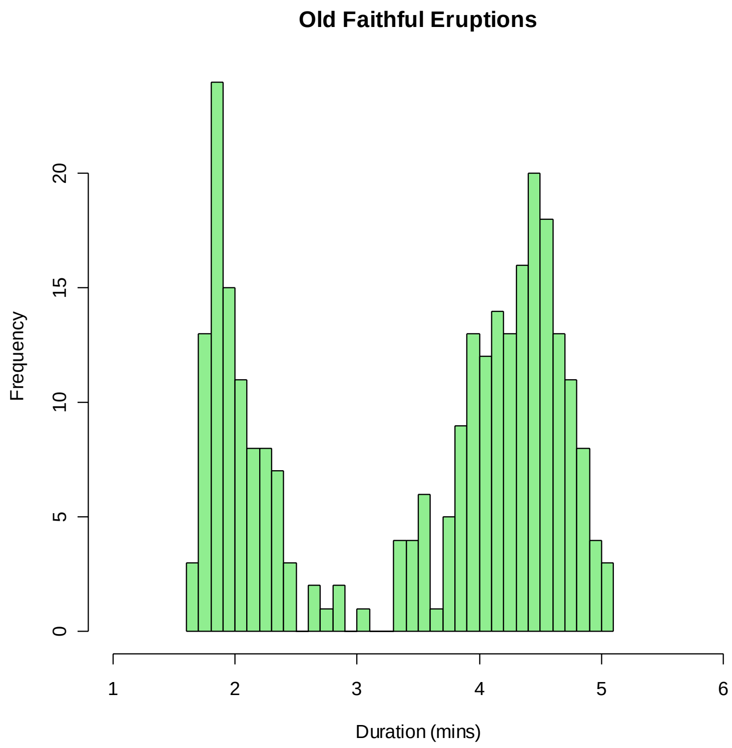
- Required about ~300 evaluations
  - Found log-likelihood of ~267.89 in 1/50 runs
  - Found log-likelihood of ~263.91 in 2/50 runs
  - Found log-likelihood of ~257.46 in 47/50 runs

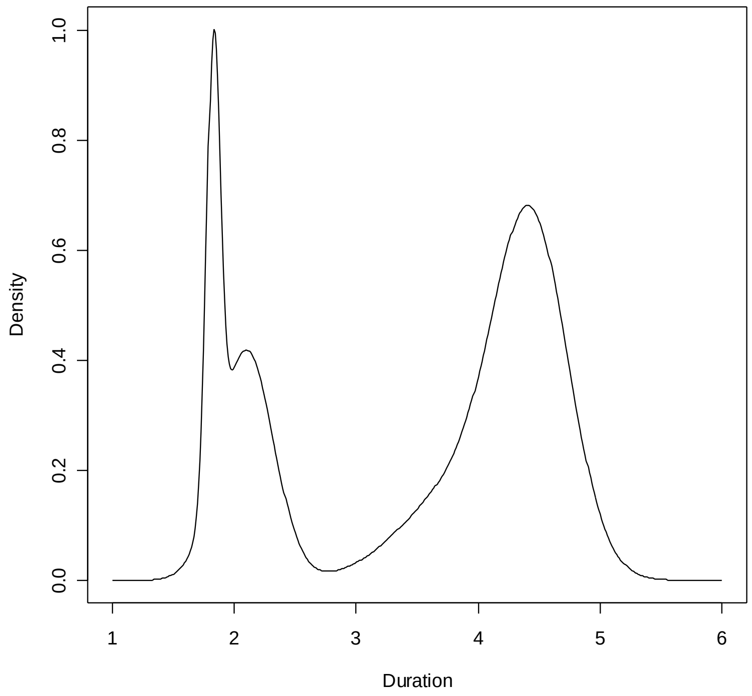
- "Appears" more reliable than with 3 components

Simulated Annealing: A Mixture of Four Normals

- Fit 11 parameters
  - 3 proportions, 4 means, 4 variances
- Required about ~100,000 evaluations
  - Found log-likelihood of ~257.46 in 50/50 runs
- Again, a grid-search in 11 dimensions would only allow ~4-5 points per dimension and find a worse solution

Four Components





Today …

- Simulated Annealing
- Markov-Chain Monte-Carlo method
- Searching for global minimum among local minima

Next Lecture

- More detailed discussion of
  - MCMC methods
  - Simulated Annealing and Probability Distributions
- Introduction to Gibbs sampling

References

- Brooks and Morgan (1995)
  Optimization using simulated annealing
  The Statistician **44:**241-257

- Kirkpatrick et al (1983)
  Optimization by simulated annealing
  Science **220:**671-680

I/O Notes for Problem Set 7

- To read data, use `"stdio.h"` library

- Functions for opening and closing files
  - `FILE * fopen(char * name, char * mode);`
  - `void fclose(FILE * f);`

- Functions for reading and writing to files
  - I recommend `fprintf` and `fscanf`
  - Analogous to `printf` and `scanf`

fopen() function

- Typical usage:

```
FILE * f = fopen("file.txt", "rt");
if (f == NULL)
    {
    printf("Error opening file\n");
    exit(1);
    }

/* Rest of code follows */
```

- File mode combines to characters:
  - "w" for writing and "r" for reading
  - "t" for text and "b" for binary

fclose()

- Makes a file available to other programs

```
fclose(f);
```

- To return to the beginning of a file use:

```
rewind(f);
```

- To check whether the end-of-file has been reached:

```
feof(f);
```

Writing to a File

- Writing a an integer and a double

```
int i = 2;
double x = 2.11;

 fprintf(f, "My secret numbers are "
            "%d and %f\n", i, x);
```

- As usual, use %d for integers, %f for doubles, %s for strings

Reading from a File

- Reading a an integer and a double

```
int i;
double x;
fscanf(f, "%d %lf\n", &i, &x);
```

- As usual, use %d for integers, `%f` for floats, `%lf` for doubles, %s for strings
- Writing `%*`*`t`* [where *`t`* is one of the type characters above] reads a value and discards it without storing.
- Returns the number of items transferred successfully

Counting Items in File

```
FILE * f = fopen(filename, "rt");
double item;
int items = 0;

// Count the number of items in file

// First skip header
fscanf(f, "%*s ");

// Read and count floating point values
// until file is exhausted
while (!feof(f) && fscanf(f, "%lf ", &item) == 1)
   items++;
```

Reading Items from a file

```
// Return to the beginning of file
rewind(f);

// Skip header again
fscanf(f, "%*s ");

// Allocate enough memory
data = alloc_vector(n = items);

// Read each item into appropriate location
for (i = 0; i < items; i++)
   fscanf(f, "%lf ", &data[i]);

// Done with this file!
fclose(f);
```