E-M algorithm : A Basic Strategy

- Complete data \((x, z)\) - what we would like to have
  - Observed data \(x\) - individual observations
  - Missing data \(z\) - hidden / missing variables

- The algorithm
  - Use estimated parameters to infer \(z\)
  - Update estimated parameters using \(x\)
  - Repeat until convergence
Recap: The E-M algorithm

Expectation step (E-step)

- Given the current estimates of parameters $\theta^{(t)}$, calculate the conditional distribution of latent variable $z$.
- Then the expected log-likelihood of data given the conditional distribution of $z$ can be obtained

$$Q(\theta|\theta^{(t)}) = \mathbb{E}_{z|x,\theta^{(t)}} \left[ \log p(x, z|\theta) \right]$$

Maximization step (M-step)

- Find the parameter that maximize the expected log-likelihood

$$\theta^{(t+1)} = \arg \max_{\theta} Q(\theta|\theta^t)$$
Summary: The E-M Algorithm

- Iterative procedure to find maximum likelihood estimate
  - E-step: Calculate the distribution of latent variables and the expected log-likelihood of the parameters given current set of parameters
  - M-step: Update the parameters based on the expected log-likelihood function

- The iteration does not decrease the marginal likelihood function
- But no guarantee that it will converge to the MLE
- Particularly useful when the likelihood is an exponential family
  - The E-step becomes the sum of expectations of sufficient statistics
  - The M-step involves maximizing a linear function, where closed form solution can often be found
Local and global optimization methods

Local optimization methods

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

- Examples
  - Golden Search
  - Nelder-Mead Simplex Method
  - E-M algorithm

Today

- Simulated Annealing
- Markov-Chain Monte-Carlo Method
- Designed to search for global minimum among many local minima
Local minimization methods

The problem

- Most minimization strategies find the nearest local minimum from the starting point
- 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
- To do so, we sometimes need to select new points that do not improve solution
- How?
Simulated Annealing

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
Simulated Annealing Strategy

- Consider decreasing series of temperatures
- For each temperature, iterate these step
  - Propose an update and evaluation 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
Local minimization methods

The problem with hill climbing is that it gets stuck on "local-maxima"

Images by Max Dama from
Global minimization with Simulated Annealing

Images by Max Dama from
Example Applications

- The traveling salesman problem (TSP)
  - Salesman must visit every city in a set
  - Given distances between pairs of cities
  - Find the shortest route through the set

- No polynomial time algorithm is known for finding optimal solution
- Simulated annealing or other stochastic optimization methods often provide near-optimal solutions.
Simulated Annealing TSP : 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
Examples of simulated annealing results
Update scheme in Simulated Annealing

- Random walk by Metropolis criterion (1953)
- Given a configuration, assume a probability proportional to the Boltzmann factor
  \[ P_A = e^{-E_A/T} \]
- Changes from \( E_0 \) to \( E_1 \) with probability
  \[ \min \left( 1, \frac{P_1}{P_0} \right) = \min \left( 1, \exp \left( - \frac{E_1 - E_0}{T} \right) \right) \]
- Given sufficient time, leads to equilibrium state
Using Markov Chains

Markov Chain Revisited

- The Markovian property

\[ \Pr(q_t | q_{t-1}, q_{t-2}, \ldots, q_0) = \Pr(q_t | q_{t-1}) \]

- Transition probability

\[ \theta_{ij} = \Pr(q_t = j | q_{t-1} = i) \]

Simulated Annealing using Markov Chain

- Start with some state \( q_t \).
- Propose a changed \( q_{t+1} \) given \( q_t \)
- Decide whether to accept change based on \( \theta_{q_t q_{t+1}} \)
  - Decision is based on relative probabilities of two outcomes
Key requirements

- Irreducibility: it is possible to get any state from any state
  - There exist $t$ where $\Pr(q_t = j | q_0 = i) > 0$ for all $(i, j)$.

- Aperiodicity: return to the original state can occur at irregular times
  \[
  \gcd\{t : \Pr(q_t = i | q_0 = i) > 0\} = 1
  \]

- These two conditions guarantee the existence of a unique equilibrium distribution
Equilibrium distribution

- Starting point does not affect results
- The marginal distribution of resulting state does not change
- Equilibrium distribution $\pi$ satisfies

$$\pi = \lim_{t \to \infty} \Theta^{t+1}$$
$$= (\lim_{t \to \infty} \Theta^t) \Theta$$
$$= \pi \Theta$$

- In Simulated Annealing, $\Pr(E) \propto e^{-E/T}$
Simulated Annealing Recipes

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
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
  - For mixture of normals, suggestion of Brooks and Morgan (1995) works well
  - Select a component to update, and sample from within plausible range
Implementing TSP: Traverse2D.h

```cpp
#ifndef __TRAVERSE_2D_H
#define __TRAVERSE_2D_H

#include <vector>
#include <algorithm>
#include <cstdlib>
#include <cmath>

class Traverse2D {
    protected:
        double distance;
        bool stale;

    public:
        std::vector<double> xs;
        std::vector<double> ys;
        std::vector<int> order;

};

#endif
```
Implementing TSP: Traverse2D.h

Traverse2D() : distance(-1), stale(true) {}

Traverse2D(std::vector<double>& _xs, std::vector<double>& _ys):
    xs(_xs), ys(_ys) {
    int n = (int)xs.size();
    if (n != ys.size()) abort();
    for(int i=0; i < n; ++i) {
        order.push_back(i);
    }
}

int numPoints() { return (int)order.size(); }

void addPoint(double x, double y) {
    xs.push_back(x);
    ys.push_back(y);
    order.push_back((int)order.size());
}
Implementing TSP: Traverse2D.h

```cpp
void initOrder() { std::sort( order.begin(), order.end() ); }

bool nextOrder() {
    stale = true;
    return std::next_permutation( order.begin(), order.end() );
}

void shuffleOrder() {
    stale = true;
    std::random_shuffle( order.begin(), order.end() );
}

void swapOrder(int x, int y) {
    stale = true;
    int tmp = order[x];
    order[x] = order[y];
    order[y] = tmp;
}
```
Implementing TSP: Traverse2D.h

double getDistance() {
    if ( stale ) {
        int n = (int)order.size();
        distance = 0;
        for(int i=1; i < n; ++i) {
            distance += ( (xs[order[i]]-xs[order[i-1]])*(xs[order[i]]-xs[order[i-1]])
                          + (ys[order[i]]-ys[order[i-1]])*(ys[order[i]]-ys[order[i-1]]) );
        }
        distance = sqrt(distance);
        stale = false;
    }
    return distance;
}

#endif // __TRAVERSE_2D_H
Implementing TSP: main()

```cpp
int main(int argc, char** argv) {
    if ( argc != 2 ) {
        std::cerr << "Usage: TSP [infile]" << std::endl;
        return -1;
    }

    Matrix615<double> xy(argv[1]);
    int n = xy.rowNums();
    if ( xy.colNums() != 2 ) {
        std::cerr << "Input matrix does not have exactly two columns" << std::endl;
        return -1;
    }

    // build graph from file
    Traverse2D graph;
    for(int i=0; i < n; ++i) {
        graph.addPoint(xy.data[i][0], xy.data[i][1]);
    }
}
```
Implementing TSP: `main()`

```cpp
int start = 0, finish = 0, nperm = 0;
double duration = 0, minDist = DBL_MAX, maxDist = 0, sumDist = 0;
std::vector<int> minOrder;
start = clock();
graph.initOrder();  // initialize order
do {
    double d = graph.getDistance();
    sumDist += d; ++nperm;
    if ( d > maxDist ) maxDist = d;
    if ( d < minDist ) {
        minDist = d;
        minOrder = graph.order;
    }
} while ( graph.nextOrder() );
finish = clock();
duration = (double)(finish-start)/CLOCKS_PER_SEC;
```
Implementing TSP: main()

```cpp
std::cout << "------------------------------------------------" << std::endl;
std::cout << "Minimum distance = " << minDist << std::endl;
std::cout << "Maximum distance = " << maxDist << std::endl;
std::cout << "Mean distance = " << sumDist/nperm << std::endl;
std::cout << "Exhaustive search duration = " << duration << " seconds" << std::endl;
std::cout << "------------------------------------------------" << std::endl;

start = clock();
runTSPSA(graph, 1e-6); // run Simulated Annealing
finish = clock();
duration = (double)(finish-start)/CLOCKS_PER_SEC;
std::cout << "SA distance = " << graph.getDistance() << std::endl;
std::cout << "SA search Duration = " << duration << " seconds" << std::endl;
std::cout << "------------------------------------------------" << std::endl;

return 0;
```
Implementing TSP: runTSPSA()

```c
#define MAX_TEMP 1000
#define N_ITER 1000

double runTSPSA(Traverse2D& graph, double eps) {
    srand(std::time(0));
    graph.shuffleOrder();

    double temperature = MAX_TEMP;
    double prevDist = graph.getDistance();
    int n = graph.numPoints();
    while( temperature > eps ) {
        for(int i=0; i < N_ITER; ++i) {
            int i1 = (int)floor( rand()/(RAND_MAX+1.) * n);
            int i2 = (int)floor( rand()/(RAND_MAX+1.) * n);
            graph.swapOrder(i1,i2);
            double newDist = graph.getDistance();
            double diffDist = newDist-prevDist;
        }
    }
}```
Implementing TSP : runTSPSA()

```java
if ( diffDist < 0 ) {
    prevDist = newDist;
}
else {
    double p = rand()/(RAND_MAX+1.);
    if ( p < exp(0-diffDist/temperature) ) {
        prevDist = newDist;
    }
    else {
        graph.swapOrder(i1,i2);
    }
}
temperature *= 0.90;
```
TSP : Working examples

```
$ cat tsp.10.in.txt
-2.30963348991357  0.0773267767084084
-1.13260001198939  0.194723763831079
-0.47887704546568  -1.49043206086804
-1.14183413926286  -0.386463669289195
-0.0684871826034848  0.362329163828058
-1.28322395967065  -0.173892955683618
-0.684913927794102  0.0967915142130205
1.87577059887638  -0.229129514295367
-0.796217725319515  1.77563911372358
0.936967861258253  -0.103803298997143
```
TSP : Working examples

$ ./TSP tsp.10.in.txt
---------------------------------------------
Minimum distance = 3.45434
Maximum distance = 8.00868
Mean distance = 6.053
Exhaustive search duration = 9.69001 seconds
---------------------------------------------
SA distance = 3.50017
SA search Duration = 0.456339 seconds
---------------------------------------------

$ ./TSP tsp.10.in.txt
---------------------------------------------
Minimum distance = 3.45434
Maximum distance = 8.00868
Mean distance = 6.053
Exhaustive search duration = 9.72787 seconds
---------------------------------------------
SA distance = 3.45434
SA search Duration = 0.457726 seconds
---------------------------------------------
TSP : Working examples

$ cat tsp.11.in.txt
-0.636066544886696 2.25053338615707
0.0860940972604061 0.231139523090642
0.219459494449743 -0.518180472158068
0.0566391380933713 -1.10184323809265
-0.300676076997908 -0.765625163407885
2.64204087640419 1.29479579271570
0.152911487506204 0.228909136397270
-0.933319389247532 -0.846940788411644
-0.447908403019059 -1.16451734926683
1.61047052169711 1.66393401261582
-1.16737084487488 1.04729096252209
TSP : Working examples

$ ./TSP tsp.11.in.txt
---------------------------------------------
Minimum distance = 3.50014
Maximum distance = 9.53825
Mean distance = 7.28444
Exhaustive search duration = 115.615 seconds
---------------------------------------------
SA distance = 3.52509
SA search Duration = 0.514433 seconds
---------------------------------------------

$ ./TSP tsp.11.in.txt
---------------------------------------------
Minimum distance = 3.50014
Maximum distance = 9.53825
Mean distance = 7.28444
Exhaustive search duration = 116.613 seconds
---------------------------------------------
SA distance = 3.50014
SA search Duration = 0.507408 seconds
---------------------------------------------
Simulated Annealing for Gaussian Mixtures

class normMixSA {
public:
    int k; // # of components
    int n; // # of data
    std::vector<double> data; // observed data
    std::vector<double> pis; // pis
    std::vector<double> means; // means
    std::vector<double> sigmas; // sds
    double llk; // current likelihood
normMixSA(std::vector<double>& _data, int _k); // constructor
void initParams(); // initialize parameters
    double updatePis(double temperature);
double updateMeans(double temperature, double lo, double hi);
double updateSigmas(double temperature, double sdlo, double sdhi);
double runSA(double eps); // run Simulated Annealing
static int acceptProposal(double current, double proposal, double temperature);
};
Evaluating Proposals in Simulated Annealing

```c
int normMixSA::acceptProposal(double current, double proposal,
                               double temperature) {
    if ( proposal < current ) return 1;  // return 1 if likelihood decreased
    if ( temperature == 0.0 ) return 0;  // return 0 if frozen
    double prob = exp(0-(proposal-current)/temperature);
    return (randu(0.,1.) < prob);  // otherwise, probabilistically accept proposal
}
```
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 or not
Updating Means

double normMixSA::updateMeans(double temperature, double min, double max) {
    int c = randn(0,k)  // select a random integer between 0..(k-1)
    double old = means[c];  // save the old mean for recovery
    means[c] = randu(min, max);  // update mean and evaluate the likelihood
    double proposal = 0-NormMix615::mixLLK(data, pis, means, sigmas);
    if ( acceptProposal(llk, proposal, temperature) ) {
        llk = proposal;  // if accepted, keep the changes
    }
    else {
        means[c] = old;  // if rejected, rollback the changes
    }
    return llk;
}
double normMixSA::updateSigmas(double temperature, double min, double max) {
    int c = randn(0,k);  // select a random integer between 0..(k-1)
    double old = sigmas[c];  // save the old mean for recovery
    sigmas[c] = randu(min, max);  // update a component and evaluate the likelihood
    double proposal = 0-NormMix615::mixLLK(data, pis, means, sigmas);
    if ( acceptProposal(llk, proposal, temperature) ) {
        llk = proposal;  // if accepted, keep the changes
    }
    else {
        sigmas[c] = old;  // if rejected, rollback the changes
    }
    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 up to 25%
  - Rescale all proportions so they sum to 1.0
Updating Mixture Proportions

double normMixSA::updatePis(double temperature) {
    std::vector<double> pisCopy = pis;  // make a copy of pi
    int c = randn(0,k);               // select a component to update
    pisCopy[c] *= randu(0.8,1.25);    // update the component
    // normalize pi
    double sum = 0.0;
    for(int i=0; i < k; ++i)
        sum += pisCopy[i];
    for(int i=0; i < k; ++i)
        pisCopy[i] /= sum;
    double proposal = 0-NormMix615::mixLLK(data, pisCopy, means, sigmas);
    if ( acceptProposal(llk, proposal, temperature) ) {
        llk = proposal;
        pis = pisCopy;  // if accepted, update pis to pisCopy
    }
    return llk;
}
Initializing parameters

```c
void normMixSA::initParams() {
    double sum = 0, sqsum = 0;
    for(int i=0; i < n; ++i) {
        sum += data[i];
        sqsum += (data[i]*data[i]);
    }
    double mean = sum/n;
    double sigma = sqrt(sqsum/n - sum*sum/n/n);
    for(int i=0; i < k; ++i) {
        pis[i] = 1./k; // uniform priors
        means[i] = data[rand()%n]; // pick random data points
        sigmas[i] = sigma; // pick uniform variance
    }
}
```
Putting things together

double normMixSA::runSA(double eps) {
    initParams(); // initialize parameter
    llk = 0-NormMix615::mixLLK(data, pis, means, sigmas); // initial likelihood
    double temperature = MAX_TEMP; // initialize temperature
    double lo = min(data), hi = max(data); // min(), max() can be implemented
    double sd = stdev(data); // stdev() can also be implemented
    double sdhi = 10.0 * sd, sdlo = 0.1 * sd;
    while ( temperature > eps ) {
        for(int i=0; i < 1000; ++i) {
            switch( randn(0,3) ) { // generate a random number between 0 and 2
                case 0: // update one of the 3\*k components
                    llk = updatePis(temperature); break;
                case 1:
                    llk = updateMeans(temperature, lo, hi); break;
                case 2:
                    llk = updateSigmas(temperature, sdlo, sdhi); break;
            }
        }
        temperature *= 0.90; // cool down slowly
    }
    return llk;
}
Running examples

user@host:~/> ./mixSimplex ./mix.dat
Minimim = 3043.46, at pi = 0.667271,
between N(-0.0304604,1.00326) and N(5.01226,0.956009)

user@host:~/> ./mixEM ./mix.dat
Minimim = -3043.46, at pi = 0.667842,
between N(-0.0299457,1.00791) and N(5.0128,0.913825)

user@host:~/> ./mixSA ./mix.dat
Minimim = 3043.46, at pi = 0.667793,
between N(-0.030148,1.00478) and N(5.01245,0.91296)
Comparisons

2-component Gaussian mixtures

- Simplex Method: 306 Evaluations
- E-M Algorithm: 12 Evaluations
- Simulated Annealing: ~100,000 Evaluations

For higher dimensional problems

- Simplex Method may not converge, or converge very slowly
- E-M Algorithm may stuck at local maxima when likelihood function is multimodal
- Simulated Annealing scale robustly with the number of dimensions.
Summary

Today - Simulated Annealing

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

Next lecture

- More on MCMC Method
- A simple Gibbs Sampler