Minimization With Parabolas

Biostatistics 615/815

Lecture 16

Last Lecture

- Root finding
  - Bisection of bracketing interval
  - Using a linear approximation
- Optimization
  - Bracketing triplet
  - Golden section search

Today ...

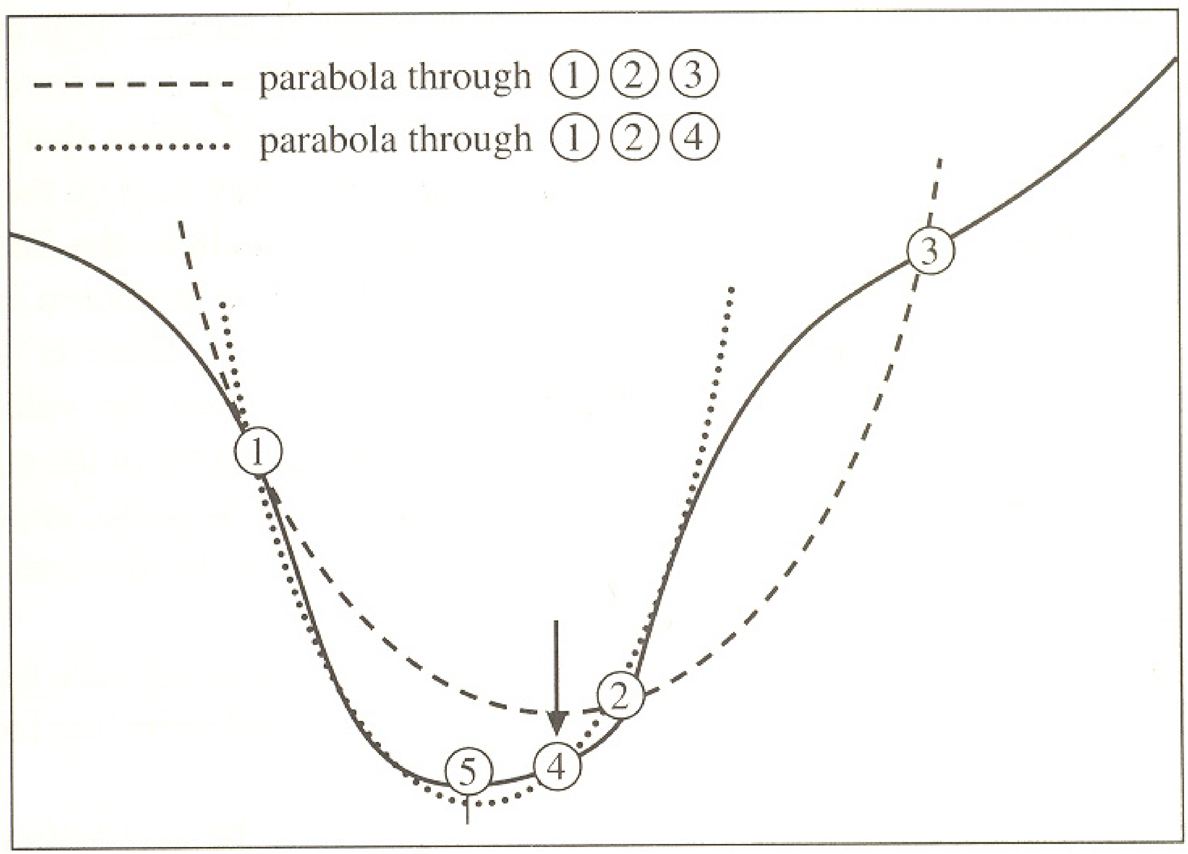
- More on numerical optimization
  - Parabolic interpolation
  - Adaptive method

- Multi-dimensional optimization problem
  - Mixture distributions

Better Numerical Optimization

- As with root finding, performance can improve substantially when a local approximation is used
- Degree of improvement depends on function being approximated
- Construct an approximation with the current bracketing triplet
    - Higher order approximations can have strange bends

Approximating The Function



Parabolic Approximation

$$f^*(x) = Ax^2 + Bx + C$$

The value which minimizes *f*(x)* is

$$x_{\min} = -\frac{B}{2A}$$

Using this strategy to minimize the function is called "inverse parabolic interpolation"

Fitting a Parabola

- Can be fitted with three points
  - Points must not be co-linear

$$C = f(x_1) - Ax_1^2 - Bx_1$$

$$B = \frac{A(x_2^2 - x_1^2) + (f(x_1) - f(x_2))}{x_1 - x_2}$$

$$A = \frac{f(x_3) - f(x_2)}{(x_3 - x_2)(x_3 - x_1)} - \frac{f(x_1) - f(x_2)}{(x_1 - x_2)(x_3 - x_1)}$$

Minimum for a Parabola

- General expression for finding minimum of a parabola fitted through three points
  - Note repeated sub-expressions

$$x_{\min} = x_2 - \frac{1}{2}\frac{(x_2 - x_1)^2\big(f(x_2) - f(x_3)\big) - (x_2 - x_3)^2\big(f(x_2) - f(x_1)\big)}{(x_2 - x_1)\big(f(x_2) - f(x_3)\big) - (x_2 - x_3)\big(f(x_2) - f(x_1)\big)}$$

Fitting a Parabola

```
// Returns the distance between b and the abscissa for the
// fitted minimum using parabolic interpolation
double parabola_step (double a, double fa,
       double b, double fb, double c, double fc)
   {
   // Quantities for placing minimum of fitted parabola
   double p = (b - a) * (fb - fc);
   double q = (b - c) * (fb - fa);
   double x = (b - c) * q - (b - a) * p;
   double y = 2.0 * (p - q);

   // Check that q is not zero
   if (fabs(y) < ZEPS)
      return golden_step (a, b, c);
   else
      return x / y;
   }
```

Caution:
Using Fitted Minimum

- Fitted minimum could overlap with one of original points
  - Could produce degenerate case
- Ensure that each new point is distinct from previously examined points

Avoiding Degenerate Steps

```
double adjust_step(double a, double b, double c, double step,
   double e)
   {
    double min_step = fabs(e * b) + ZEPS;

   if (fabs(step) < min_step);
      return step > 0 ? min_step : -min_step;

   // If the step ends up to close to previous points,
   // return zero to force a golden ratio step ...
   if (fabs(b + step - a) <= e || fabs(b + step - c) <= e)
      return 0.0;

   return step;
   }
```

Generating New Points

- Use parabolic interpolation by default
- Check whether improvement is slow
  - Steps are not rapidly decreasing in length
- Switch to golden section if function is uncooperative

Calculating Step Size

```
double calculate_step(double a, double fa,
        double b, double fb, double c, double fc,
        double last_step, double e)
   {
   double step = parabola_step(a, fa, b, fb, c, fc);
   step = adjust_step(a, b, c, step, e);

   if (fabs(step) > fabs(0.5 * last_step) || step == 0.0)
      step = golden_step(a, b, c);

   return step;
   }
```

Overall

- The main function simply has to:
  - Generate new points using building blocks
  - Update the triplet bracketing the minimum
  - Check for convergence

Overall Minimization Routine

```
double find_minimum(double (*func)(double), double a, double b, double c,
                   double e)
   {
   double fa = (*func)(a), fb = (*func)(b), fc = (*func)(c);
   double step1 = (c - a) * 0.5, step2 = (c - a) * 0.5;

   while ( fabs(c - a) > fabs(b * e) + ZEPS)
      {
      double step = calculate_step (a, fa, b, fb, c, fc, step2, e);
      double x = b + step; double fx = (*func)(x);

      if (fx < fb)
         { if (x > b) { a = b; fa = fb; } else { c = b; fc = fb; }
           b = x; fb = fx; }
      else
         if (x < b) { a = x; fa = fx; } else { c = x; fc = fx; }

      step2 = step1; step1 = step;
      }
   return b;
   }
```

Important Characteristics

- Parabolic interpolation often convergences faster
  - The preferred algorithm
- Golden search provides performance guarantee
  - A fall-back for uncooperative functions
- Switch algorithms when convergence slow
  - Allow parabolic interpolation one poor choice
- Avoid testing points that are too close

Brent's Strategy

- Most popular strategy for minimization without derivatives
  - Part of Richard Brent's PhD thesis in 1971

- Similar to the one we described:
  - Inverse Quadratic Interpolation, where possible
  - Golden Section Search, fall-back

Brent's Strategy

- Track 6 points
  - Not all distinct
  - The bracket boundaries (a, b)
  - The current minimum (x)
  - The second and third smallest values (w, v)
  - The new point to be examined (u)

- Parabolic interpolation uses (x, w, v) to propose new value for u

Recommended Reading

- Numerical Recipes in C (or C++)
  - Press, Teukolsky, Vetterling, Flannery
  - Chapters 10.0 – 10.2
- Excellent resource for scientific computing
- Online at
  - http://www.numerical-recipes.com/
  - http://www.library.cornell.edu/nr/

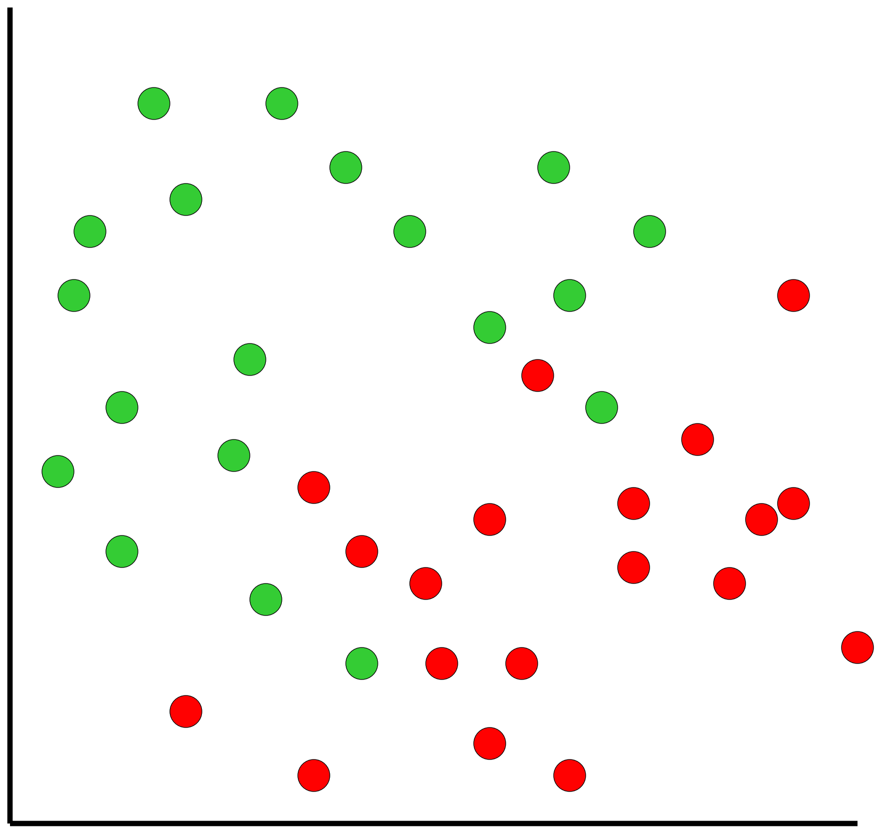
Next topic: Multi-dimensional Optimization

- Simplex method of Nelder and Mead
- The Expectation Maximization algorithm
- Monte-Carlo Methods
  - Metropolis algorithm
  - Gibbs sampling

A Multi-Dimensional Problem: Mixture Distributions

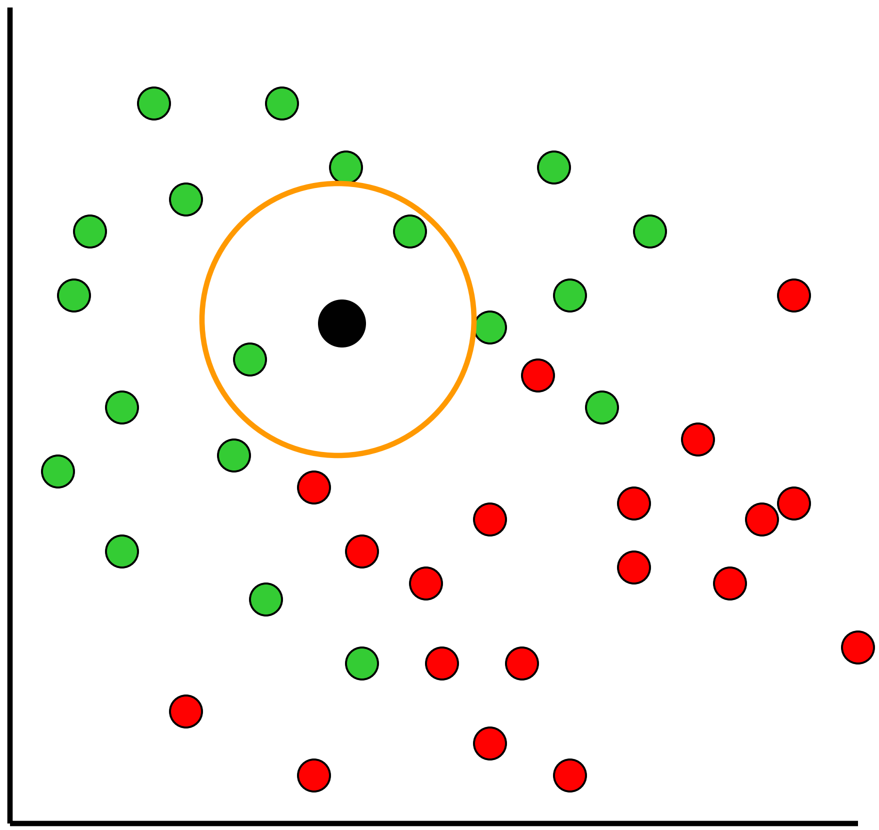
- Interesting application for multidimensional optimization
- Related to many useful statistical problems
  - Clustering
  - Classification

Classification



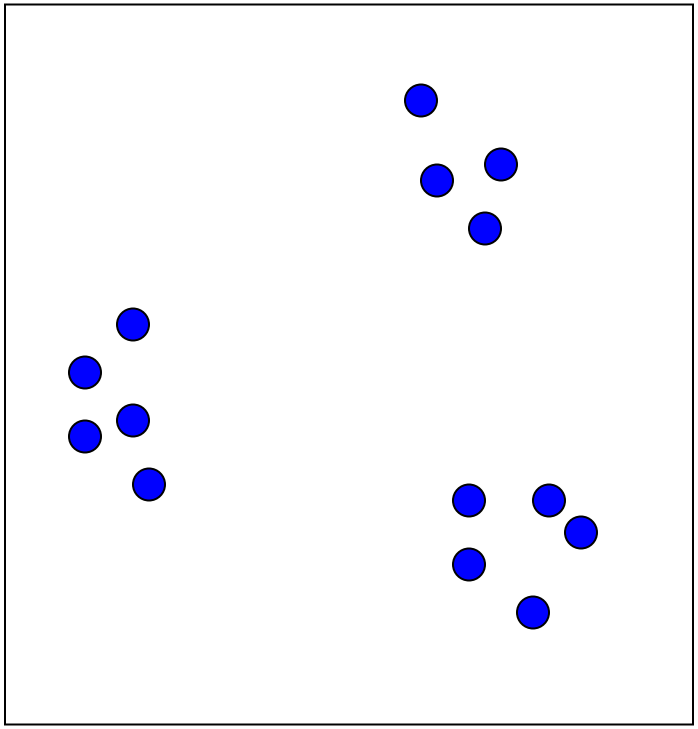
- Given elements with known groupings …

Classification



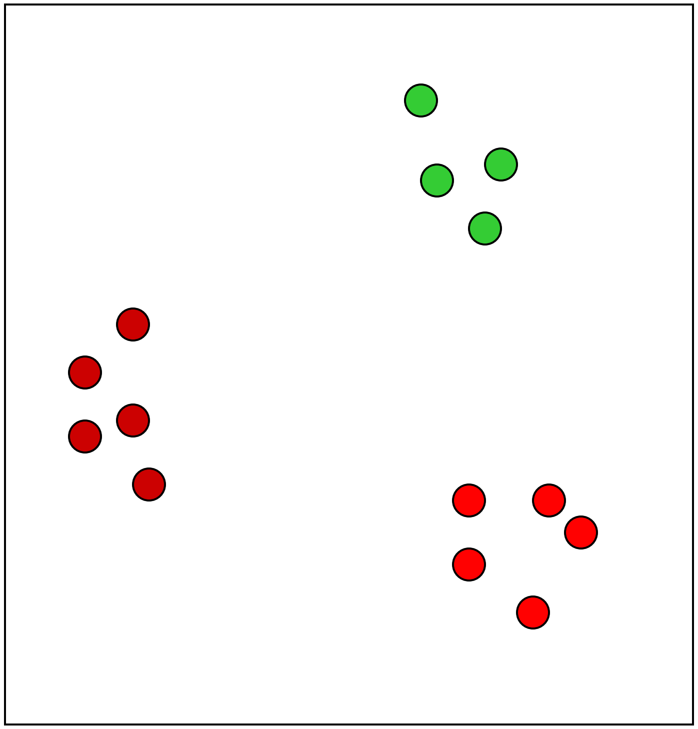
- Given elements with known groupings …
- Assign grouping for a new element

Clustering



- Starting with points with unknown sources
  …

Clustering

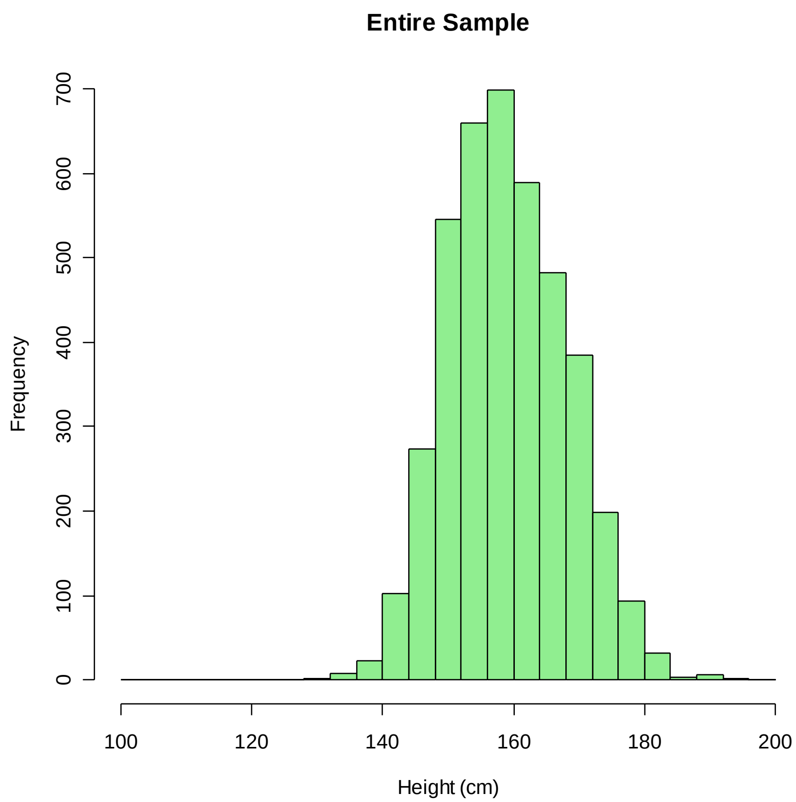


- Starting with points with unknown sources
- Find appropriate grouping scheme

A simple distribution

- For many continuous measurements, normal distribution is a good starting point
- Parameters are easy to estimate from the sample

Heights for 4,102 Individuals



Mean = ~160 cm, Variance = ~80

Normal Density

- If the data is normally distributed, the density function for each component is

$$f(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

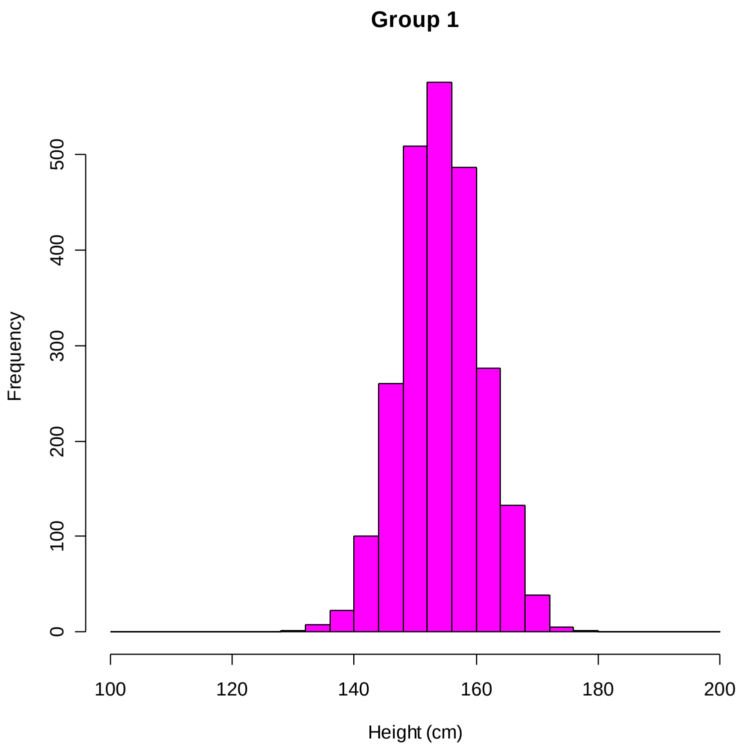
C Code:
Normal Density

```
#include <math.h>

double square(double x)
   { return x * x; }

double dnorm(double x, double mu, double sigma)
   {
   return 1.0 / (sigma * sqrt(M_PI * 2.0)) *
          exp (-0.5 * square((x - mu)/sigma));
   }
```

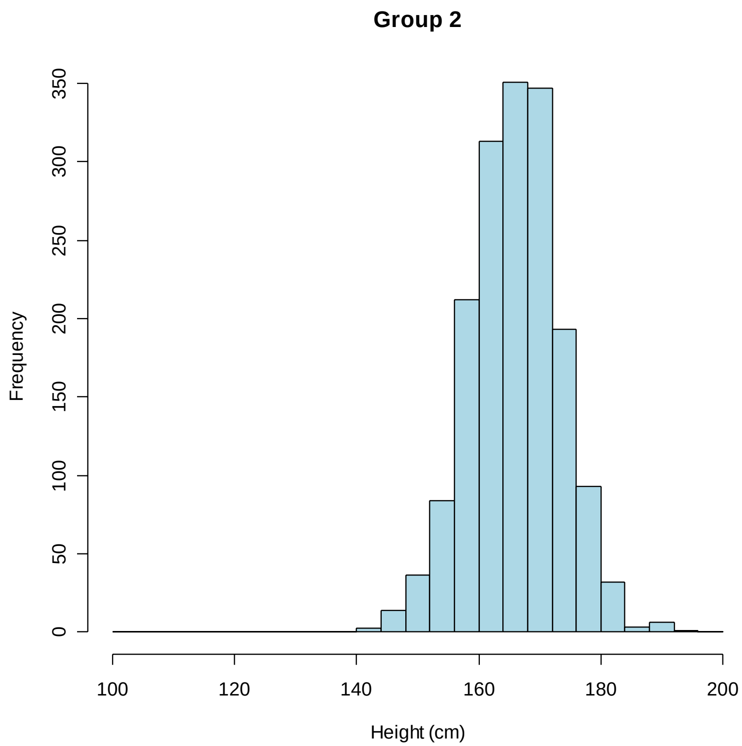
A Simple Mixture Distribution

- Observations are univariate
  - Single measurement
- Each component has a normal distribution

Two Underlying Distributions



Mean = ~154 cm



Mean = ~166 cm

A General Mixture Distribution

$$p(x \mid \boldsymbol{\pi}, \boldsymbol{\varphi}, \eta) = \pi_1 f(x \mid \varphi_1, \eta) + \ldots + \pi_k f(x \mid \varphi_2, \eta)$$

- x is the observation
- π are the mixture proportions
- f is the probability density function
- ϕ are parameters for each component
- η are parameters shared among components
- k is the number of components

C Code: Mixture Distribution

```
double dmix(double x,
            int k, double probs[], double means[],
            double sigmas[])
   {
   int i;
   double density = 0.0;

   for (i = 0; i < k; i++)
      density += probs[i] *
                 dnorm(x, means[i], sigmas[i]);

   return density;
   }
```

Maximum Likelihood Approach

- Find the parameters that maximize the likelihood for the entire sample

$$L = \prod_j p(x_j \mid \pi, \varphi, \eta)$$

- It is advisable to consider the log-likelihood instead to avoid underflows!

$$\ell = \sum_j \log p(x_j \mid \pi, \varphi, \eta)$$

C Code:
Overall Log-Likelihood

```
double mixLLK(int n, double x[],
              int k, double probs[], double means[],
              double sigmas[])
   {
   int i;
   double llk = 0.0;

   for (int i = 0; i < n; i++)
      llk += log(dmix(x[i], k, probs, means, sigmas));

   return llk;
   }
```

Missing Data Formulation

- For each observation i, we are missing some specific (and interesting) information

- The group membership indicator Z_i
  - If this were observed, the entire problem could become quite simple

Classification Probabilities

$$\Pr(Z_j = i \mid \boldsymbol{\pi}, \boldsymbol{\varphi}, \eta) = \pi_i$$

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

- Results from the application of Bayes' theorem
- Probabilistic interpretation…

C Code: Classification Probabilities

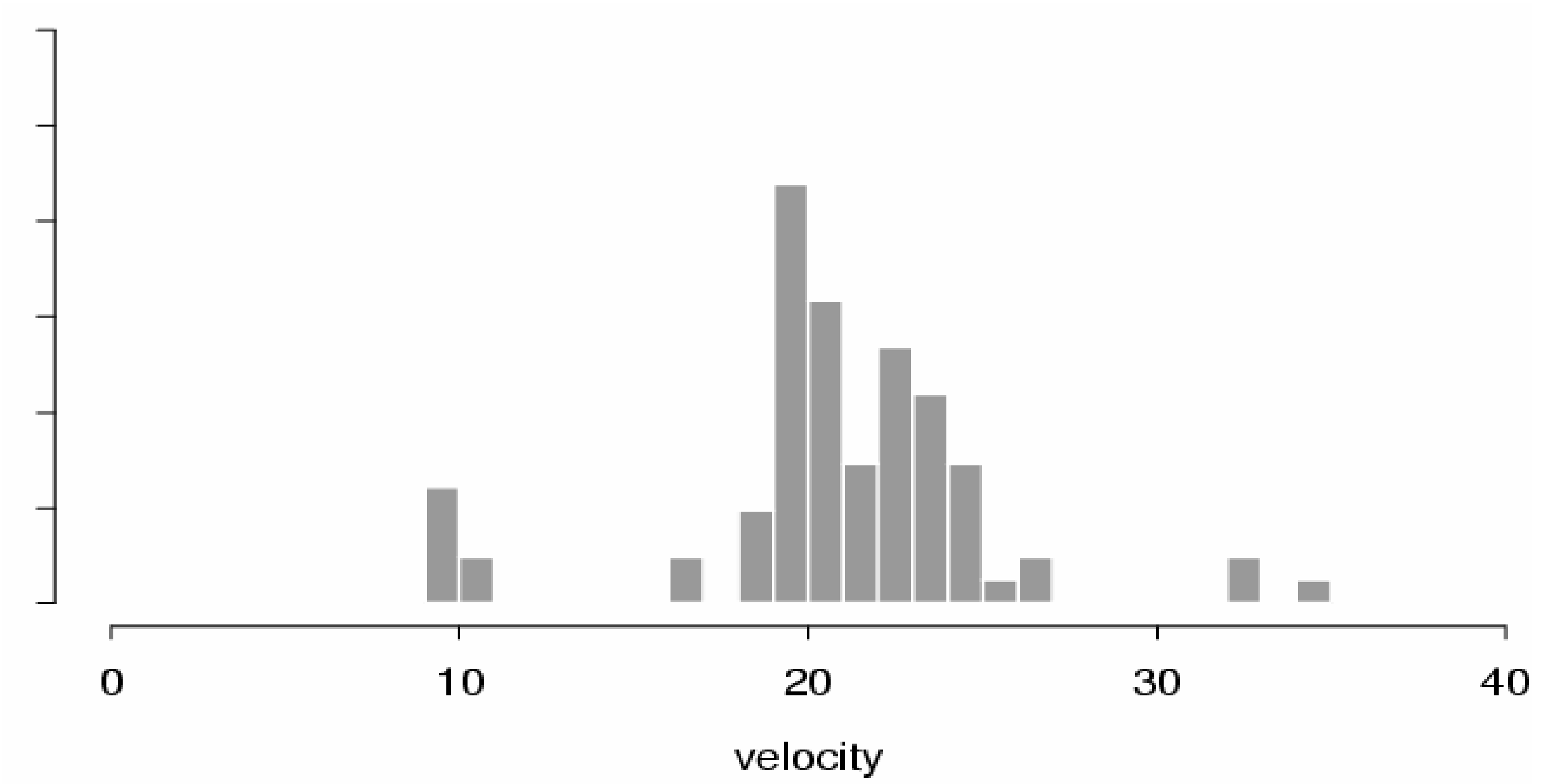
```
double classprob(int j, double x, int k,
                 double probs[], double means[],
                 double sigmas[])
   {
   double p = probs[j] *
              dnorm(x, means[j], sigmas[j]);

   return p / dmix(x, k, probs, means, sigmas);
   }
```

- Calculates the probability that observation *x* belongs to component *j*

A related problem

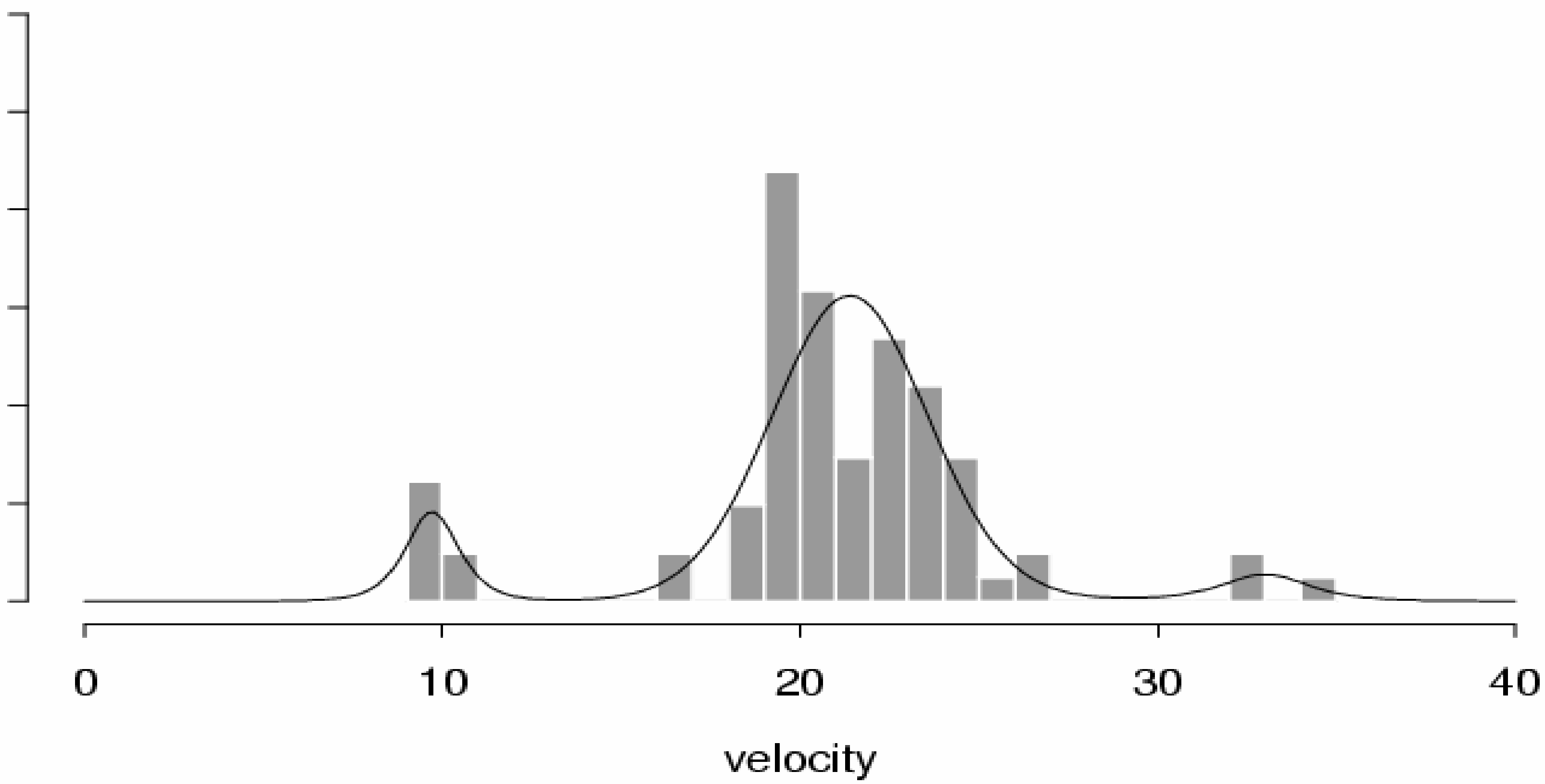
- Estimating the number of components
  - Can be interesting in itself!
- The maximum likelihood approach requires a preset number of components
- Penalized likelihood approaches required…

Example: Galaxy Speeds

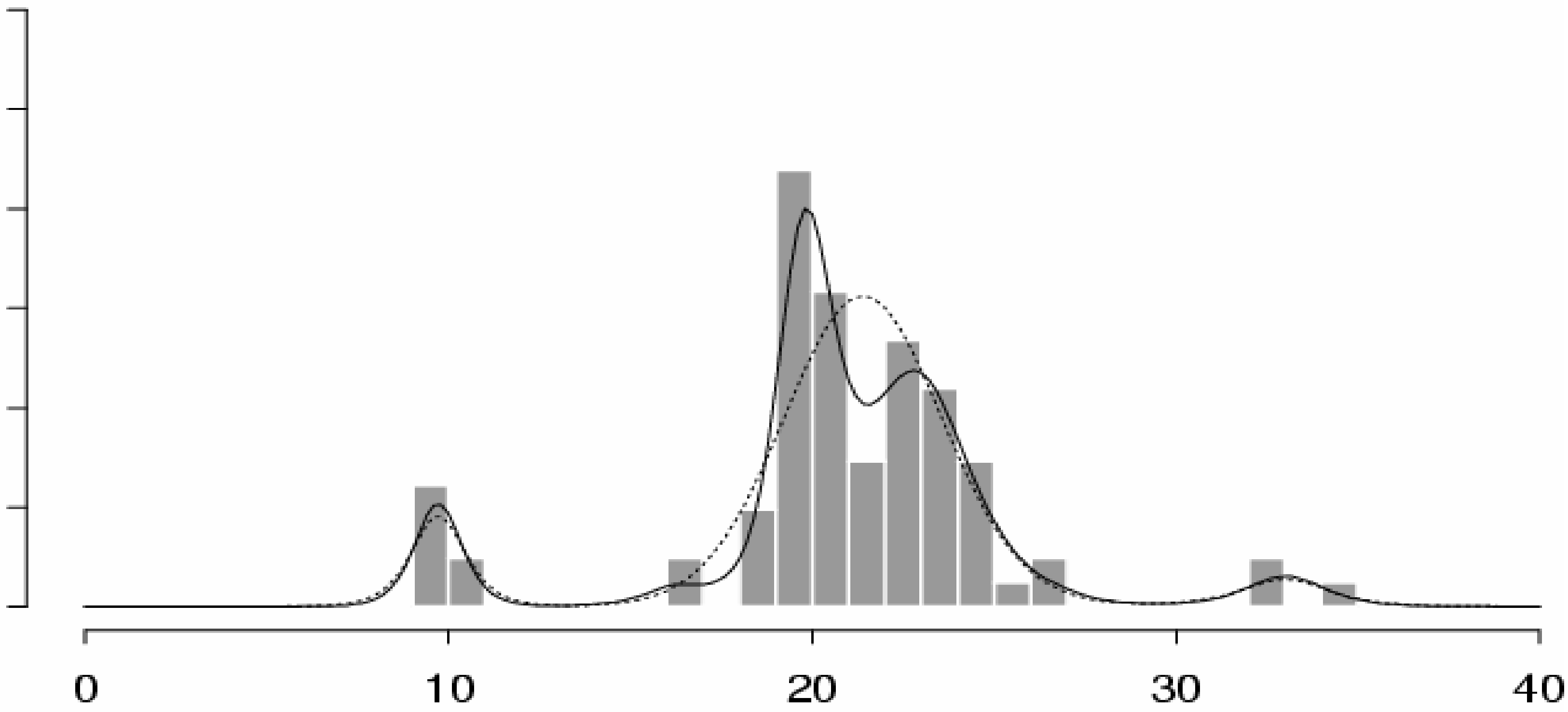


Data of Postman et al. (1986) in the Astronomical Journal.

Fitting 3 Components (Stephens, 1997)



Fitting 6 Components (Stephens, 1997)



Today …

- An introduction to mixture distributions

- Basic routines for modeling these data

- In the upcoming lectures, we will examine how to fit these mixtures appropriately to data

Additional Reading

- If you need a refresher on mixture distributions…
  - Bayesian Methods for Mixture Distributions
    M. Stephens (1997)
    http://www.stat.washington.edu/stephens/
  - Chapter 1 recommended