Computational Statistics with Application to Bioinformatics

Prof. William H. Press
Spring Term, 2008
The University of Texas at Austin

Unit 11: Gaussian Mixture Models and EM Methods

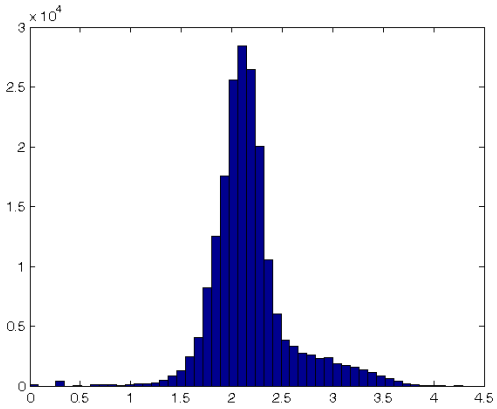
Unit 11: Gaussian Mixture Models and EM Methods (Summary)

- Gaussian Mixture Models (GMMs)
  - multidimensional PDF is sum of Gaussians
  - each has own mean (location) and covariance (shape)
  - model consists of the means, covariances, and a probabilistic assignment of every data point to the Gaussians
- E step: If we knew the Gaussians, we could assign the points
  - by relative probability density of each Gaussian at each point
- M step: If we knew the assignment, we could estimate the Gaussians
  - by weighted means of the points assigned to each of them
- EM method applied to GMMs: alternate between E and M steps
  - start with randomly chosen points
  - use logarithms and log-sum-exp formula
- See one- and two-dimensional examples
  - ideal cases (simulations from Gaussians) recover correct parameters
  - non-ideal cases (real data) is not Gaussian
    - we get decent multi-dimensional approximations of the data
    - more components give better approximation
    - but there is no particular meaning to any one component
- Variations of GMMs
  - "k-means clustering" is GMM for dummies
- General theory of EM methods
  - Jensen's inequality applied to log function
  - repeatedly maximize a two-argument bounding function
  - often a powerful method for missing data problems

Gaussian Mixture Models (GMMs)

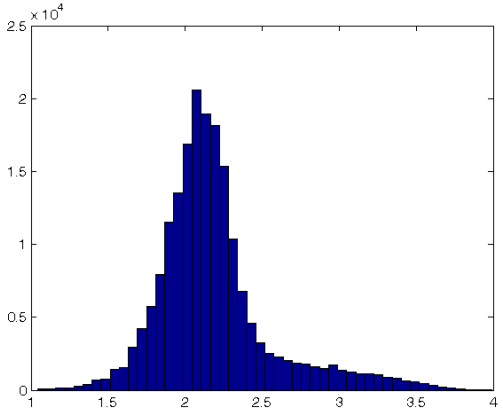
- A method for fitting multiple Gaussians to (possibly) a set of multi-dimensional data points
  - properties of Gaussians are used in detail: doesn't easily generalize to other fitting functions
- Also exemplifies Expectation Maximization (EM) methods
  - an important class of methods (Dempster, Laird, & Rubin)
  - we'll see other EM methods later
- Let's first try it in 1-D on our favorite (somewhat ill-posed) data set

```
g = readgenestats('genestats.dat');
exons = cell2mat(g.exonlen);
hist(log10(exons),50)
```




```
data = log10(exons(log10(exons)>1
& log10(exons)<4));
hist(data,50)
```

(Let's trim the outliers.)



Key to the notational thicket:

M dimensions
$k = 1 \ldots K$ Gaussians
$n = 1 \ldots N$ data points
$P(k)$ population fraction in k
$P(\mathbf{x}_n)$ model probability at $\mathbf{x}_n$

$\boldsymbol{\mu}_k$ (the K means, each a vector of length M)
$\boldsymbol{\Sigma}_k$ (the K covariance matrices, each of size $M \times M$)
$P(k|n) \equiv p_{nk}$ (the K probabilities for each of N data points)

"probabilistic assignment" of a data point to a component!

$$\mathcal{L} = \prod_n P(\mathbf{x}_n)$$ overall likelihood of the model

$$P(\mathbf{x}_n) = \sum_k N(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) P(k)$$ specify the model as a sum of Gaussians

$$N(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{M/2} \det(\boldsymbol{\Sigma})^{1/2}} \exp[-\tfrac{1}{2}(\mathbf{x} - \boldsymbol{\mu}) \cdot \boldsymbol{\Sigma}^{-1} \cdot (\mathbf{x} - \boldsymbol{\mu})]$$

Expectation, or E-step: suppose we know the model, but not the assignment of individual points.

(so called because it's probabilistic assignment by expectation value)

$$p_{nk} \equiv P(k|n) = \frac{N(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) P(k)}{P(\mathbf{x}_n)}$$

Maximization, or M-step: suppose we know the assignment of individual points, but not the model.

$$\hat{\boldsymbol{\mu}}_k = \sum_n p_{nk} \mathbf{x}_n \Big/ \sum_n p_{nk}$$

$$\hat{\boldsymbol{\Sigma}}_k = \sum_n p_{nk} (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k) \otimes (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k) \Big/ \sum_n p_{nk}$$

$$\hat{P}(k) = \frac{1}{N} \sum_n p_{nk}$$

(so called because [theorem!] the overall likelihood increases at each step)

- Can be proved that alternating E and M steps converges to (at least a local) maximum of overall likelihood
- Convergence is sometimes slow, with long "plateaus"
- Often start with k randomly chosen data points as starting means, and equal (usually spherical) covariance matrices
  - but then had better try multiple re-starts

Because Gaussians underflow so easily, a couple of tricks are important:

1) Use logarithms!

$$\log N(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})\cdot\boldsymbol{\Sigma}^{-1}\cdot(\mathbf{x}-\boldsymbol{\mu}) - \frac{M}{2}\log(2\pi) - \frac{1}{2}\log\det(\boldsymbol{\Sigma})$$

2) Do the sum $$P(\mathbf{x}_n) = \sum_k N(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) P(k)$$

by the "log-sum-exp" formula:

$$\log\left(\sum_i \exp(z_i)\right) = z_{\max} + \log\left(\sum_i \exp(z_i - z_{\max})\right)$$

We'll skip these tricks for our 1-D example, but use them (via NR3) in multidimensional examples.

The E-step in 1-D looks like this:

```
mu = [2. 3.];
sig = [0.2 0.4];
pr = @(x) exp(-0.5*((x-mu)./sig).^2)./sig;
pr(2.5)
ans =
    0.2197    1.1446

prn = @(x) pr(x)./sum(pr(x));
prn(2.5)
ans =
    0.1610    0.8390

prns = zeros([numel(data),2]);
for j=1:numel(data); prns(j,:)=prn(data(j)); end;
prns(100:110,:)
ans =
    0.9632    0.0368
    0.0803    0.9197
    0.7806    0.2194
    0.6635    0.3365
    0.5819    0.4181
    0.9450    0.0550
    0.9801    0.0199
    0.8824    0.1176
    0.9703    0.0297
    0.9661    0.0339
    0.7806    0.2194
```

Probability of a single component. Don't need to get the π's right, since will only look at relative probabilities.

Normalized probability.

Compute for all the points (show only 10).

The M-step in 1-D looks like this:

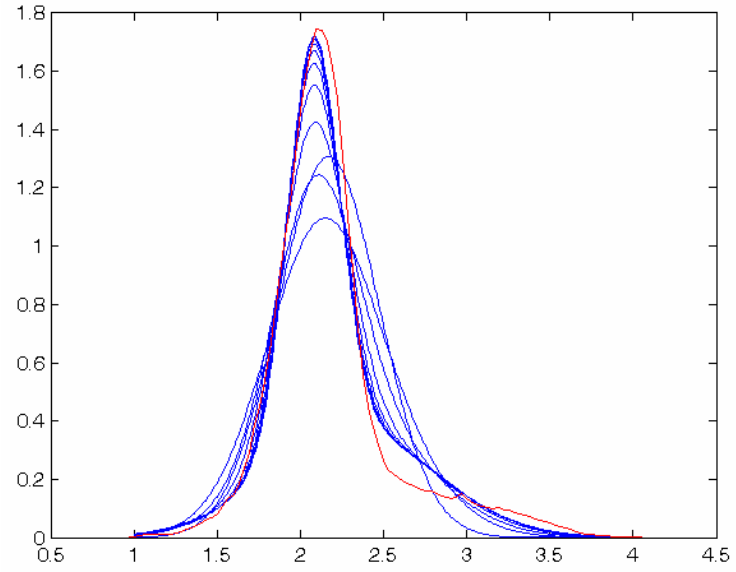
```
mu = sum(prns.*repmat(data,[1,2]), 1) ./ sum(prns,1)
xmmu = repmat(data,[1,2]) - repmat(mu,[numel(data),1]);
sig = sqrt(sum(prns .* xmmu.^2, 1) ./ sum(prns,1))
pop = sum(prns,1)/numel(data)
```

(Elegant in Matlab's data-parallel language. But, unfortunately, doesn't generalize well to multidimensions. We'll use NR3 instead, which also includes the tricks already mentioned.)

Let's show 10 iterations

```
mu = [randsample(data,1) randsample(data,1)]
sig = [.3 .3]
for jj=1:10,
    pr = @(x) exp(-0.5*((x-mu)./sig).^2)./(2.506*sig);
    prn = @(x) pr(x)./sum(pr(x));
    for j=1:numel(data); prns(j,:)=prn(data(j)); end;
    mu = sum(prns.*repmat(data,[1,2]), 1) ./ sum(prns,1);
    xmmu = repmat(data,[1,2]) - repmat(mu,[numel(data),1]);
    sig = sqrt(sum(prns .* xmmu.^2, 1) ./ sum(prns,1));
    pop = sum(prns,1)/numel(data);
    thefunc = @(x) sum(pop.*pr(x),2);
    x = 1:.01:4;
    f = arrayfun(thefunc,x);
    plot(x,f,'b');
    hold on;
end;
[f x] = ksdensity(data);
plot(x,f,'r')
hold off;
```

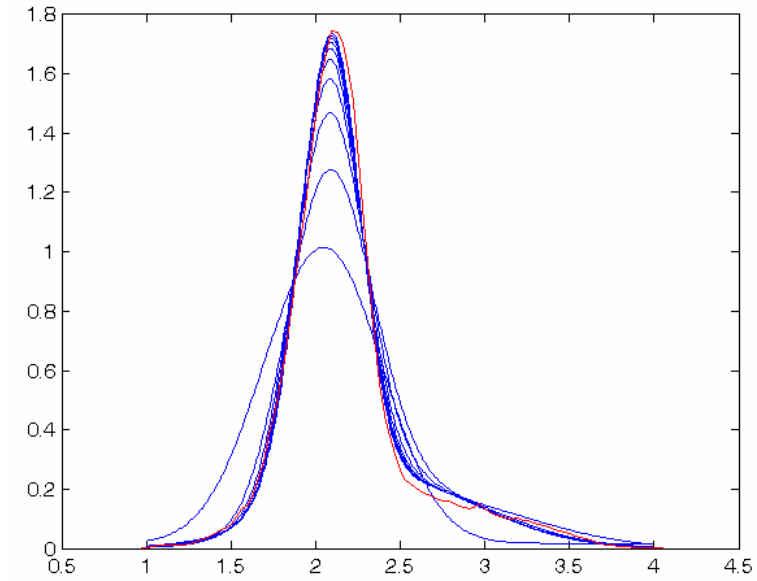
Matlab has "kernel smoothing density estimate"
(we could have used IQagent, e.g.)



2 components

```
mu =
    2.0806    2.3100
sig =
    0.1545    0.5025
pop =
    0.5397    0.4603
```

Notice that this makes a different set of "compromises" from other fitting methods. It *hates* having points in regions of "zero" probability and would rather tolerate only fair fits in the "shoulders". It is not the same as weighted LS to binned data!



3 components

```
mu =
    2.1278    2.0260    2.4186
sig =
    0.1515    0.1892    0.5451
pop =
    0.3403    0.3399    0.3198
```

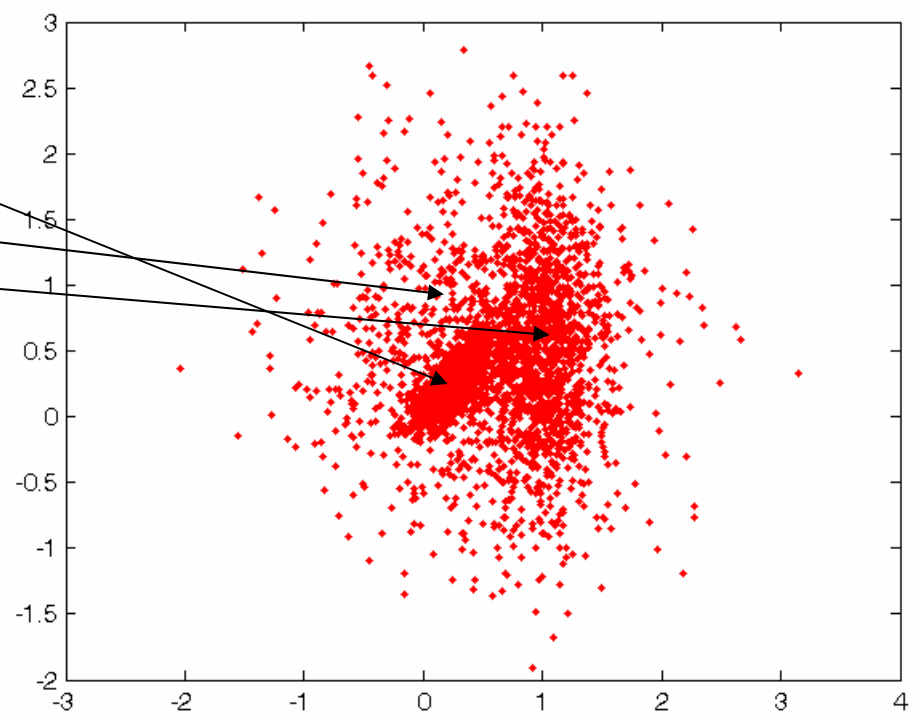
More components will converge to an excellent approximation. This does *not* mean that the components mean anything physically!

In this example, almost all starting points give the same, presumably global, max likelihood.

Let's move to 2 dimensions and do an "ideal", then a "non-ideal", example.

Ideal: we generate Gaussians, then, we fit to Gaussians

```
mu1 = [.3 .3];
sig1 = [.04 .03; .03 .04];
mu2 = [.5 .5];
sig2 = [.5 0; 0 .5];
mu3 = [1 .5];
sig3 = [.05 0; 0 .5];
rsamp = [mvnrnd(mu1,sig1,1000); ...
         mvnrnd(mu2,sig2,1000); ...
         mvnrnd(mu3,sig3,1000)];
size(rsamp)
ans =
        3000           2
plot(rsamp(:,1),rsamp(:,2),'.r')
```



Example of making an NR3 class available to Matlab:

```
#include "stdafx.h"
#include "..\nr3_matlab.h"
#include "cholesky.h"
#include "gaumixmod.h"

/* Matlab usage:
     gmm('construct',data,means)
     loglike = gmm('step',nsteps)
     [mean sig] = gmm(k)
     resp = gmm('response')
     gmm('delete')
*/

Gaumixmod *gmm = NULL;

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {
     int i,j,nn,kk,mm;
     if (gmm) {nn=gmm->nn; kk=gmm->kk; mm=gmm->mm;}
     if (gmm && nrhs == 1 && mxT(prhs[0]) == mxT<Doub>() ) { // [mean sig] = gmm(k)
          Int k = Int( mxScalar<Doub>(prhs[0]) );
          if (nlhs > 0) {
               VecDoub mean(mm,plhs[0]);
               for (i=0;i<mm;i++) mean[i] = gmm->means[k-1][i]; // k argument comes in 1-based
          }
          if (nlhs > 1) {
               MatDoub sig(mm,mm,plhs[1]);
               for (i=0;i<mm;i++) for (j=0;j<mm;j++) sig[i][j] = gmm->sig[k-1][i][j];
          }
     } else if (nrhs == 1 && mxScalar<char>(prhs[0]) == 'd') {        // gmm('delete')
          delete gmm;
```

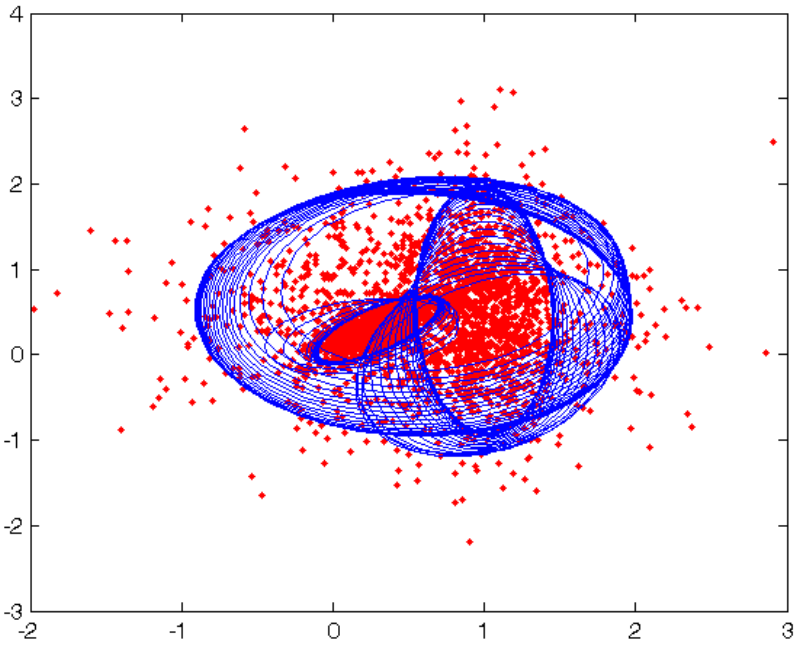
This is the key: it defines some functions and constructors for easy interfacing to Matlab lhs's and rhs's (shown in red). See http://nr.com/nr3_matlab.html for documentation.

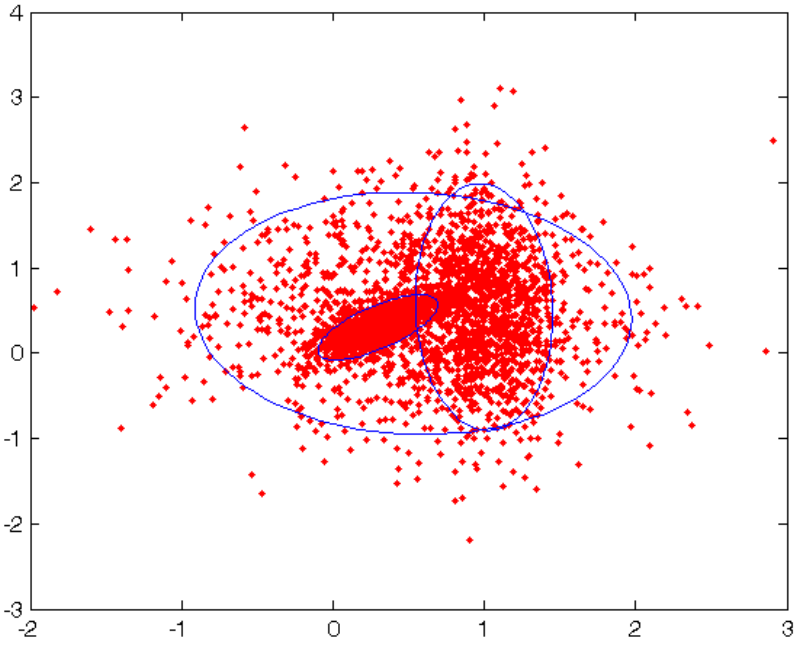
```
    } else if (gmm && nrhs == 1 && mxScalar<char>(prhs[0]) == 'r') { // gmm('response')
        if (nlhs > 0) {
            MatDoub resp(nn,kk,plhs[0]);
            for (i=0;i<nn;i++) for (j=0;j<kk;j++) resp[i][j] = gmm->resp[i][j];
        }
    } else if (gmm && nrhs == 2 && mxT(prhs[1]) == mxT<Doub>()) {
      // deltaloglike = gmm('step',nsteps)
        Int nstep = Int(mxScalar<Doub>(prhs[1]));
        Doub tmp;
        for (i=0;i<nstep;i++) {
            tmp = gmm->estep();
            gmm->mstep();
        }
        if (nlhs > 0) {
            Doub &deltaloglike = mxScalar<Doub>(plhs[0]);
            deltaloglike = tmp;
        }
    } else if (nrhs == 3 && mxT(prhs[0]) == mxT<char>()) { // gmm('construct',data,means)
        MatDoub data(prhs[1]), means(prhs[2]);
        if (means.ncols() != data.ncols()) throw("wrong dims in gmm 1");
        if (means.nrows() >= data.nrows()) throw("wrong dims in gmm 2");
        // user probably didn't transpose
        if (gmm) delete gmm;
        gmm = new Gaumixmod(data,means);
    } else {
        throw("bad call to gmm");
    }
    return;
}
```

```
gmm('construct',rsamp',means');
deltaloglike = 1.e10
while deltaloglike > 0.1;
    deltaloglike = gmm('step',1)
    for k=1:3;
        [mmu ssig] = gmm(k);
        [x y] = errorellipse(mmu',ssig',2,100);
        plot(x,y,'b');
    end;
end;
```

Note the transposes. Transpose everything going in and coming out, since Matlab has Fortran, not C, storage order.

remember our errorellipse function?

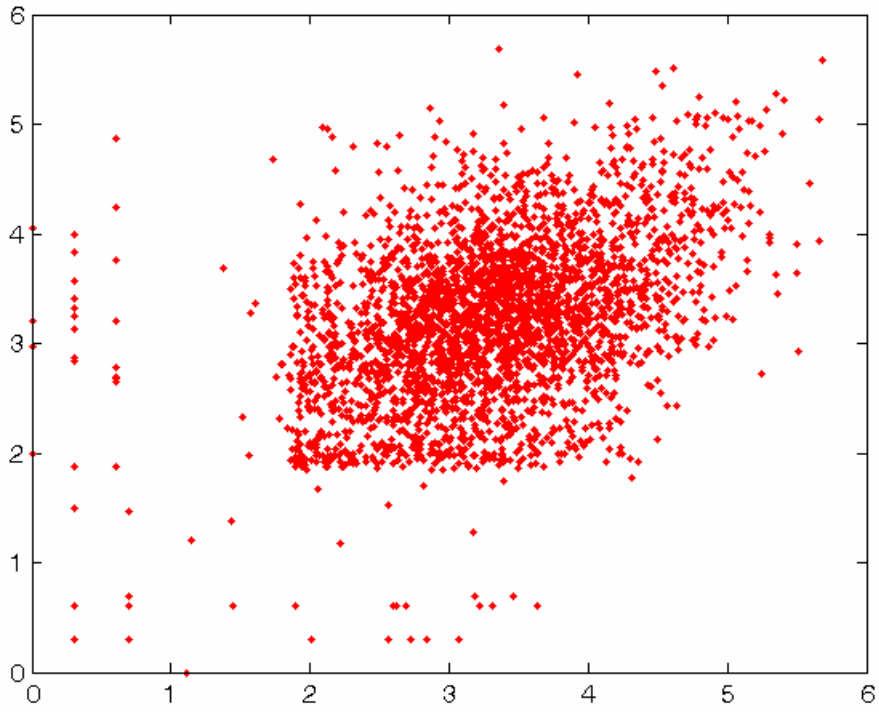




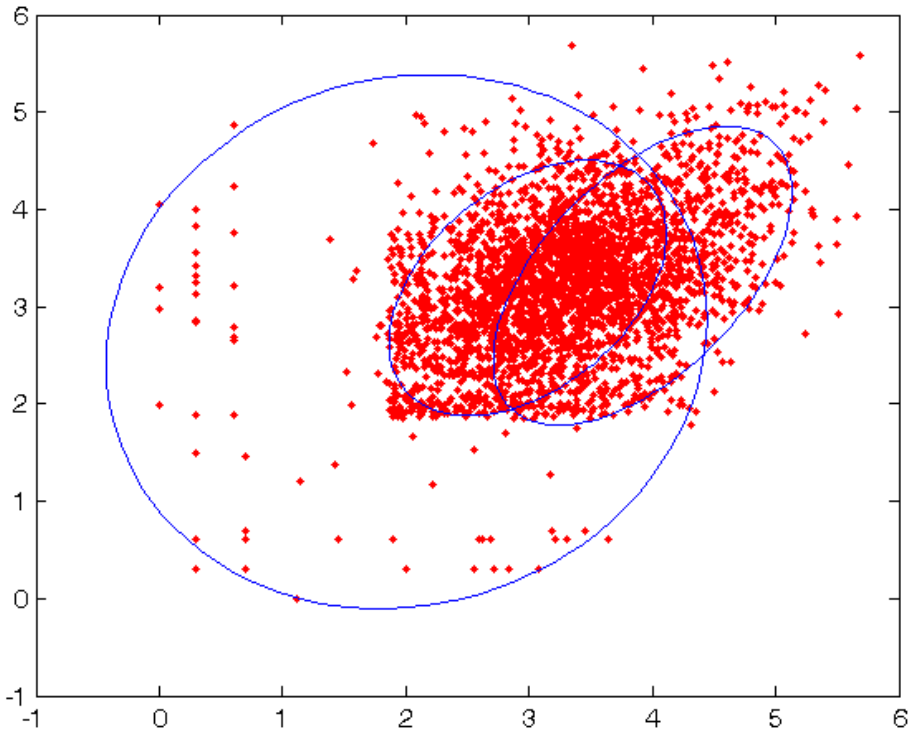
This “ideal” example converges rapidly to the right answer.

For a non-ideal example, let's go back to our data on 1st and 2nd exon log-lengths. In 2-dimensions, we can easily see that something non-GMM is going on! For the general problem in >2 dimensions, it's often hard to visualize whether this is the case or not, so GMMs get used "blindly".

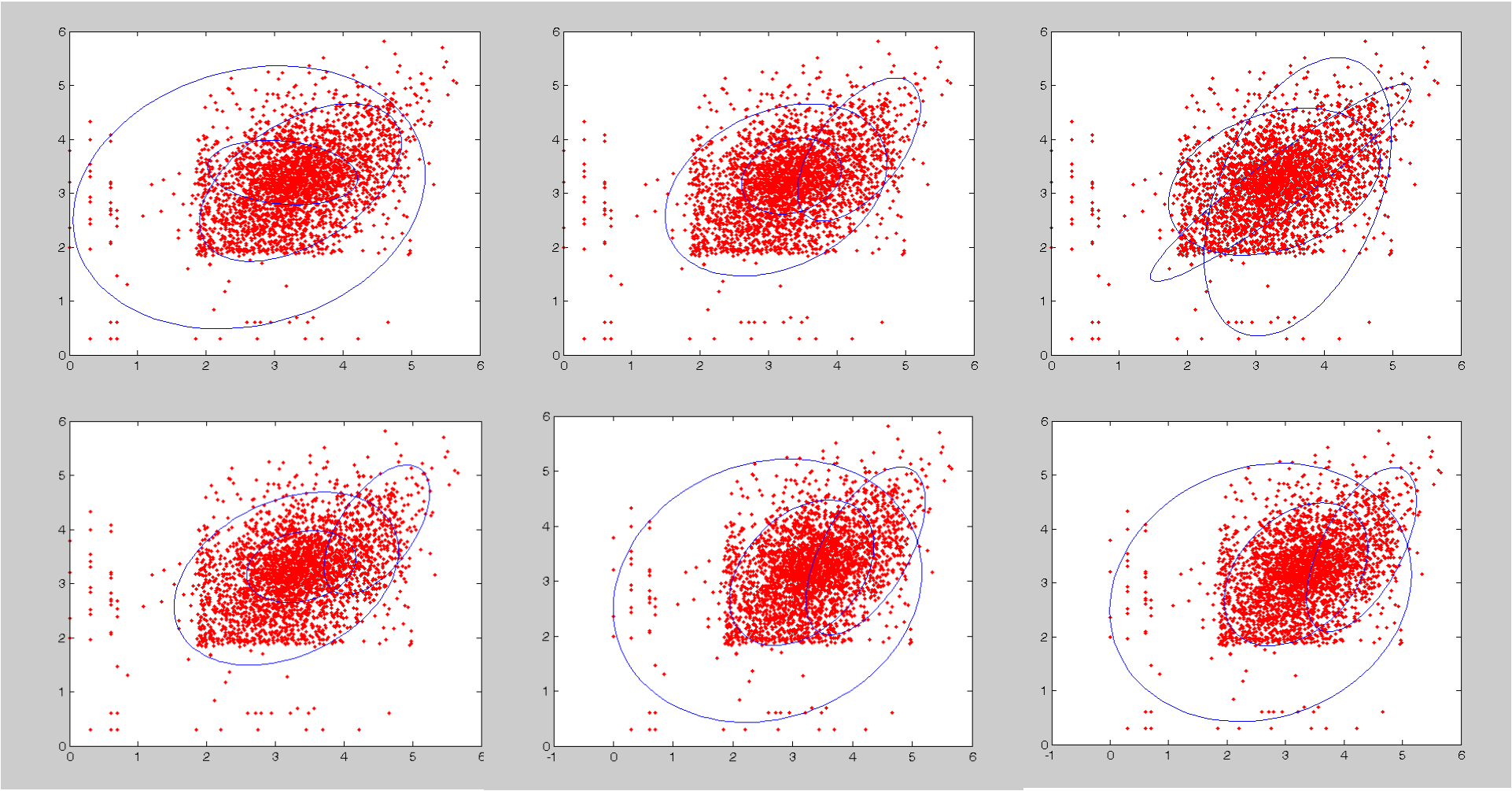
```
g = readgenestats('genestats.dat');
ggg = g(g.ne>2,:);
which = randsample(size(ggg,1),3000);
iilen = ggg.intronlen(which);
i1len = zeros(size(which));
i2len = zeros(size(which));
for j=1:numel(i1len), i1llen(j) = log10(iilen{j}(1)); end;
for j=1:numel(i2len), i2llen(j) = log10(iilen{j}(2)); end;
plot(i1llen,i2llen,'.r')
hold on
rsamp = [i1llen', i2llen'];
size(rsamp)
ans =
        3000           2
```



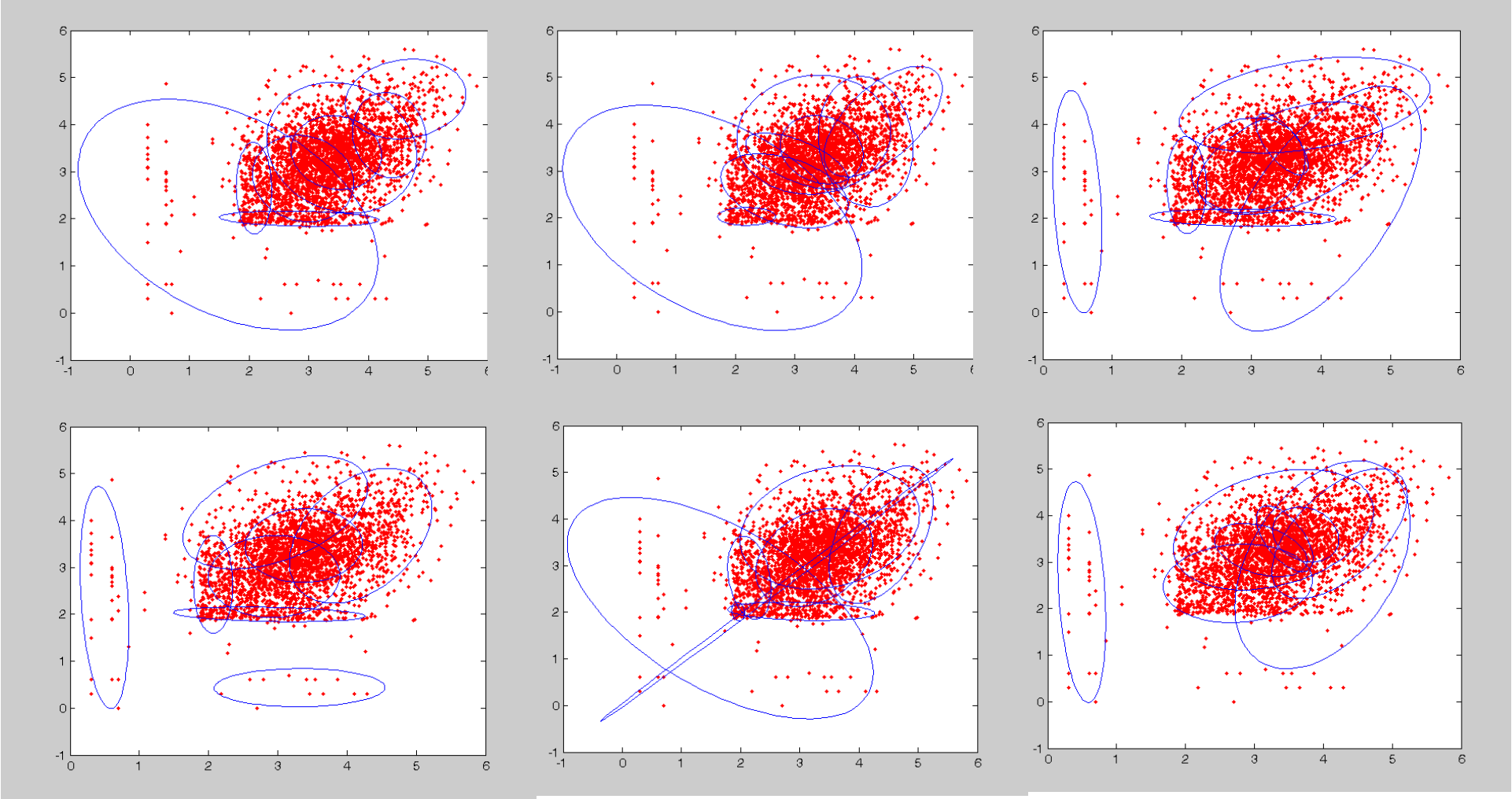
```
ncomp = 3;
plot(rsamp(:,1),rsamp(:,2),'.r')
hold on
means = zeros(ncomp,2);
for k=1:ncomp; means(k,:) = rsamp(ceil(rand*3000),:); end;
gmm('construct',rsamp',means');
deltaloglike = 1.e10;
while deltaloglike > 0.1;
    deltaloglike = gmm('step',1);
end;
for k=1:ncomp;
    [mmu ssig] = gmm(k);
    [x y] = errorellipse(mmu',ssig',2,100);
    plot(x,y,'b');
end;
hold off
```



We don’t always land on the same local maximum, although there seem to be just a handfull.



Eight components:



The ones with higher likelihood are pretty good as summaries of the data distribution (absent a predictive model). But the individual components are unstable and have little or no meaning. "Fit a lot of Gaussians for interpolation, but don't believe them."

Variations on the theme of GMMs:

- You can constrain the Σ matrices to be diagonal
  - when you have reason to believe that the components individually have no cross-correlations (align with the axes)

$$(\hat{\boldsymbol{\Sigma}}_k)_{mm} = \sum_n p_{nk}[(\mathbf{x}_n)_m - (\hat{\boldsymbol{\mu}}_k)_m]^2 \Big/ \sum_n p_{nk}$$

- Or constrain them to be multiples of the unit matrix
  - make all components spherical

$$(\hat{\boldsymbol{\Sigma}}_k) = \mathbf{1} \times \left(\sum_n p_{nk} |\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k|^2 \Big/ \sum_n p_{nk} \right)$$

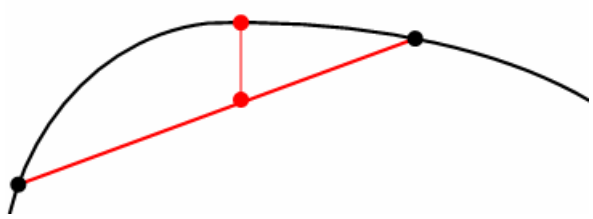
- Or fix $\Sigma = \varepsilon \mathbf{1}$ (infinitesimal times unit matrix)
  - don't re-estimate Σ, only re-estimate μ
  - this assigns points 100% to the closest cluster (so don't actually need to compute any Gaussians, just compute distances)
  - it is called "K-means clustering"
    - kind of GMM for dummies
    - widely used (there are a lot of dummies!)
    - probably always better to use spherical GMM (middle bullet above)

Let's look at the theory behind EM methods more generally:

Preliminary: Jensen's inequality

If a function is concave (downward), then

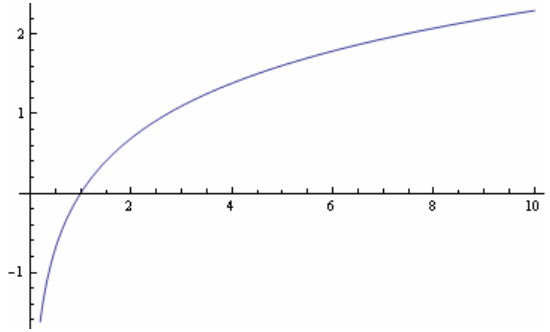
function(interpolation) ≥ interpolation(function)



Log is concave (downward). Jensen's inequality is thus:

$$\text{If} \qquad \sum_i \lambda_i = 1$$

$$\text{Then} \qquad \ln \sum_i \lambda_i Q_i \geq \sum_i \lambda_i \ln Q_i$$



This gets used a lot when playing with log-likelihoods. Proof of the EM method that we now give is just one example.

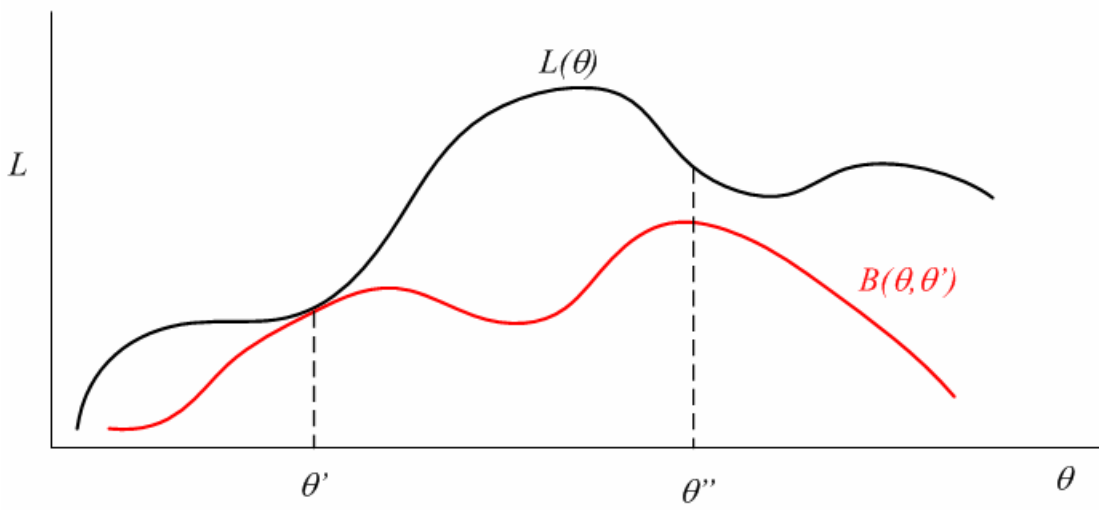
The basic EM theorem:

$\mathbf{x}$ are the data
$\mathbf{z}$ are missing data or nuisance variables
$\boldsymbol{\theta}$ are parameters to be determined

Find $\boldsymbol{\theta}$ that maximizes the log-likelihood of the data:

$$
\begin{aligned}
L(\boldsymbol{\theta}) &\equiv \ln P(\mathbf{x}|\boldsymbol{\theta}) \\
&= \ln \left[\sum_{\mathbf{z}} P(\mathbf{x}|\mathbf{z}\boldsymbol{\theta}) P(\mathbf{z}|\boldsymbol{\theta}) \right] \qquad \text{marginalize over z} \\
&= \ln \left[\sum_{\mathbf{z}} P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}') \frac{P(\mathbf{x}|\mathbf{z}\boldsymbol{\theta}) P(\mathbf{z}|\boldsymbol{\theta})}{P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}')} \right] - \ln P(\mathbf{x}|\boldsymbol{\theta}') + L(\boldsymbol{\theta}') \\
&\geq \sum_{\mathbf{z}} P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}') \ln \left[\frac{P(\mathbf{x}|\mathbf{z}\boldsymbol{\theta}) P(\mathbf{z}|\boldsymbol{\theta})}{P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}') P(\mathbf{x}|\boldsymbol{\theta}')} \right] + L(\boldsymbol{\theta}') \\
&= \sum_{\mathbf{z}} P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}') \ln \left[\frac{P(\mathbf{x}\mathbf{z}|\boldsymbol{\theta})}{P(\mathbf{z}\mathbf{x}|\boldsymbol{\theta}')} \right] + L(\boldsymbol{\theta}') \\
&\equiv B(\boldsymbol{\theta}, \boldsymbol{\theta}') \quad \text{for any } \boldsymbol{\theta}', \text{ a bound on } L(\boldsymbol{\theta})
\end{aligned}
$$

Notice that at $\boldsymbol{\theta} = \boldsymbol{\theta}'$ we have $L(\boldsymbol{\theta}) = L(\boldsymbol{\theta}')$,
so the bound touches the actual likelihood:



So, if we maximize $B(\boldsymbol{\theta}, \boldsymbol{\theta}')$ over $\boldsymbol{\theta}$, we are guaranteed that the new max $\boldsymbol{\theta}''$ will increase $L(\boldsymbol{\theta})$. This can terminate only by converging to (at least a local) max of $L(\boldsymbol{\theta})$ (Can you see why?)

And it works whether the maximization step is local or global.

So the general EM algorithm repeats the maximization:

$$\boldsymbol{\theta}'' = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\mathbf{z}} P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}') \ln \left[\frac{P(\mathbf{xz}|\boldsymbol{\theta})}{P(\mathbf{zx}|\boldsymbol{\theta}')} \right]$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\mathbf{z}} P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}') \ln \left[P(\mathbf{xz}|\boldsymbol{\theta}) \right]$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{\mathbf{z}} P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}') \ln \left[P(\mathbf{x}|\mathbf{z}\boldsymbol{\theta}) P(\mathbf{z}|\boldsymbol{\theta}) \right]$$

each form is sometimes useful

sometimes (missing data) no dependence on z

sometimes (nuisance parameters) a uniform prior

computing this is the E-step

This is an expectation that can often be computed in some better way than literally integrating over all possible values of z.

maximizing this is the M-step

This is a general way of handling missing data or nuisance parameters if you can estimate the probability of what is missing, given what you see (and a parameters guess).

Might not be instantly obvious how GMM fits this paradigm!

$\mathbf{z}$ (missing) is the assignment of data points to components
$\boldsymbol{\theta}$ consists of the $\boldsymbol{\mu}$s and $\boldsymbol{\Sigma}$s

$$P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}') \to p_{nk}$$

$$\sum_{\mathbf{z}} P(\mathbf{z}|\mathbf{x}\boldsymbol{\theta}') \ln\left[P(\mathbf{x}|\boldsymbol{\theta})\right] \to -\sum_{n,k} p_{nk}\left[(\mathbf{x}_n - \boldsymbol{\mu}_k)\cdot\boldsymbol{\Sigma}_k^{-1}\cdot(\mathbf{x}_n - \boldsymbol{\mu}_k) - \ln\det\boldsymbol{\Sigma}_k\right]$$

Showing that this is maximized by the previous re-estimation formulas for μ and Σ is a multidimensional (fancy) form of the theorem that the mean is the measure of central tendency that minimizes the mean square deviation.

See Wikipedia: Expectation-Maximization Algorithm for detailed derivation.

The next time we see an EM method will be when we discuss Hidden Markov Models. The "Baum-Welch re-estimation algorithm" for HMMs is an EM method.