CS395T
Computational Statistics with Application to Bioinformatics

Prof. William H. Press
Spring Term, 2011
The University of Texas at Austin

Lecture 16

Key to the notational thicket:

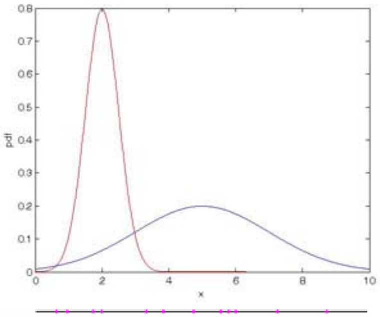
M dimensions

$k = 1 \ldots K$ Gaussians “components”

$n = 1 \ldots N$ 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

$$P(\mathbf{x}_n) = \sum_k N(\mathbf{x}_n \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) P(k)$$

specify the model as a mixture of Gaussians

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

Goal is to find *all* of the above, starting with only the $\mathbf{x}_n$

(So far this could be frequentist or Bayesian, although it was invented by frequentists.)

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
```

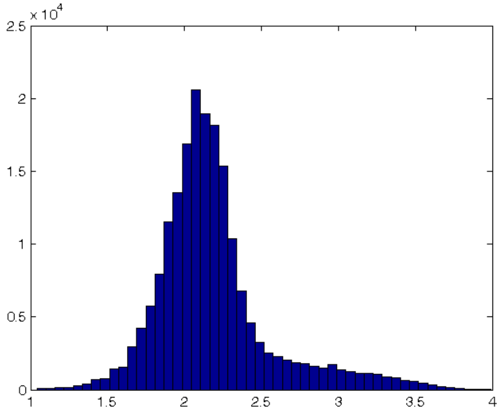
Probabilities of each component. Don’t need to get the normalizing π’s right, since will (Bayes) normalize across components...

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

…like this. Normalized probability.

```
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
```

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)
```

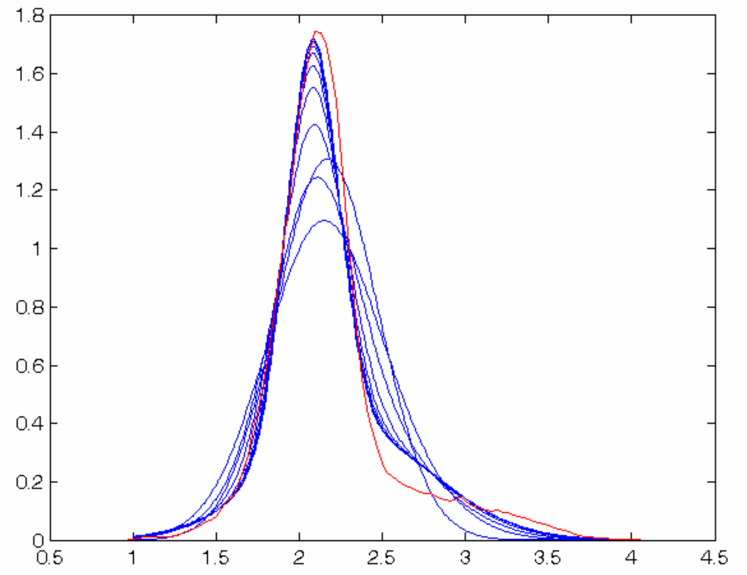
CSEM prelims alert: You should be able to understand and write code like this in MATLAB.

(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;
```

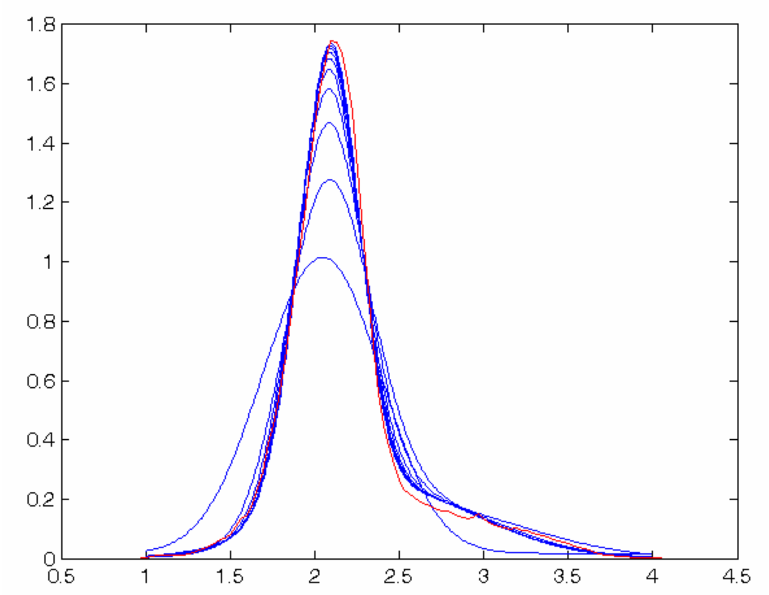
Matlab has "kernel smoothing density estimate" that is convenient for plotting the actual data as a smooth curve. (But I wouldn't trust it further than that!)



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

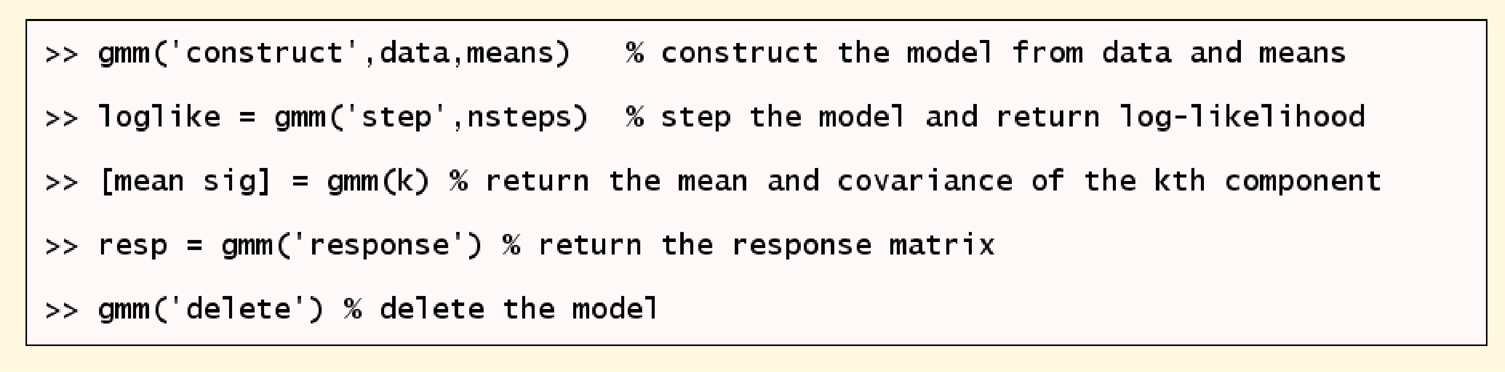
```
mu =
    2.1278    2.0260    2.4186
sig =
    0.1515    0.1892    0.5451
pop =
    0.3403    0.3399    0.3198
```

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

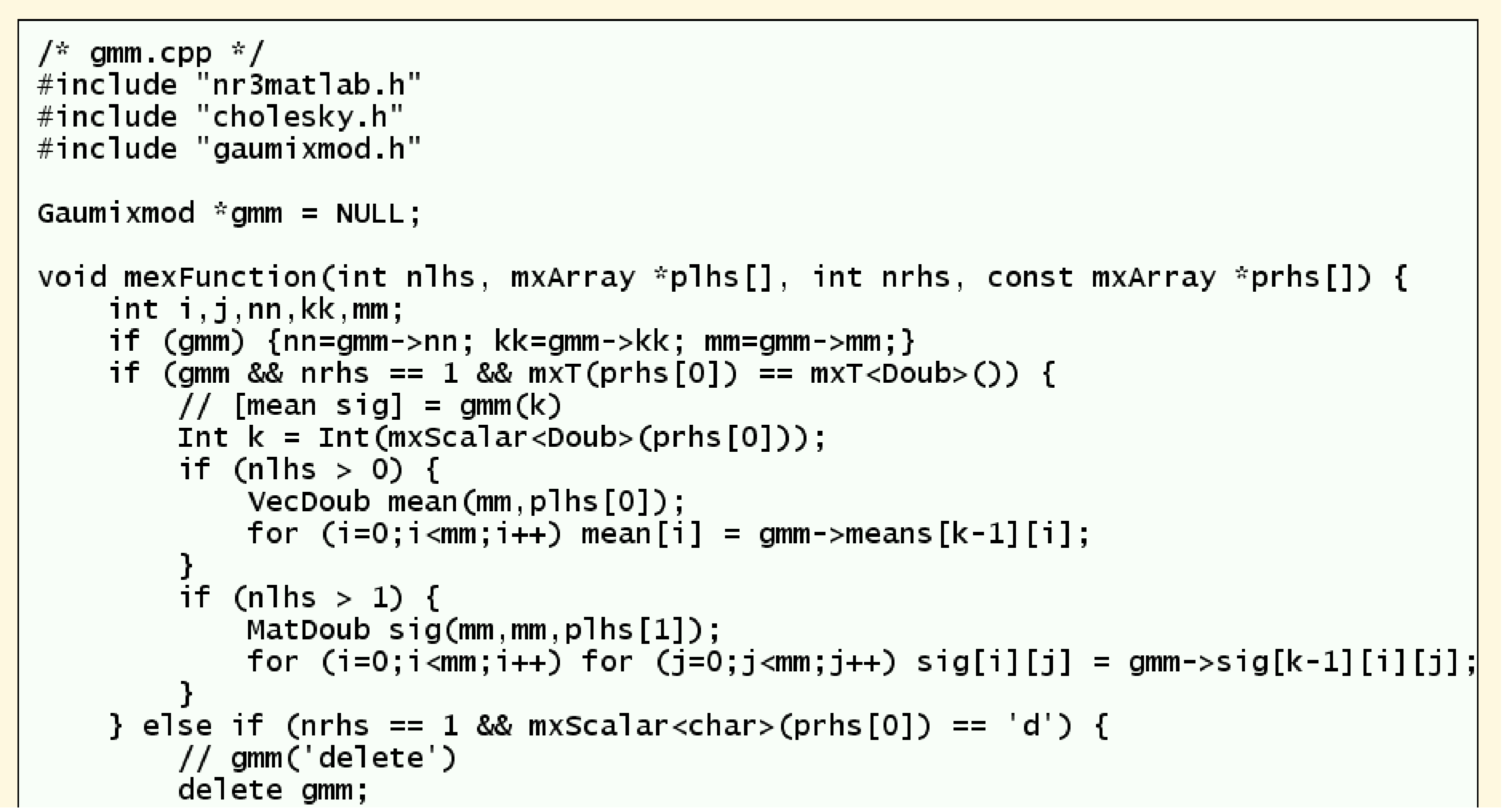
In this example, almost all starting points give the same, presumably global, max likelihood.

from: http://www.nr.com/nr3_matlab.html

MATLAB doesn’t have a GMM routine, but NR3 does, and it can be harnessed:

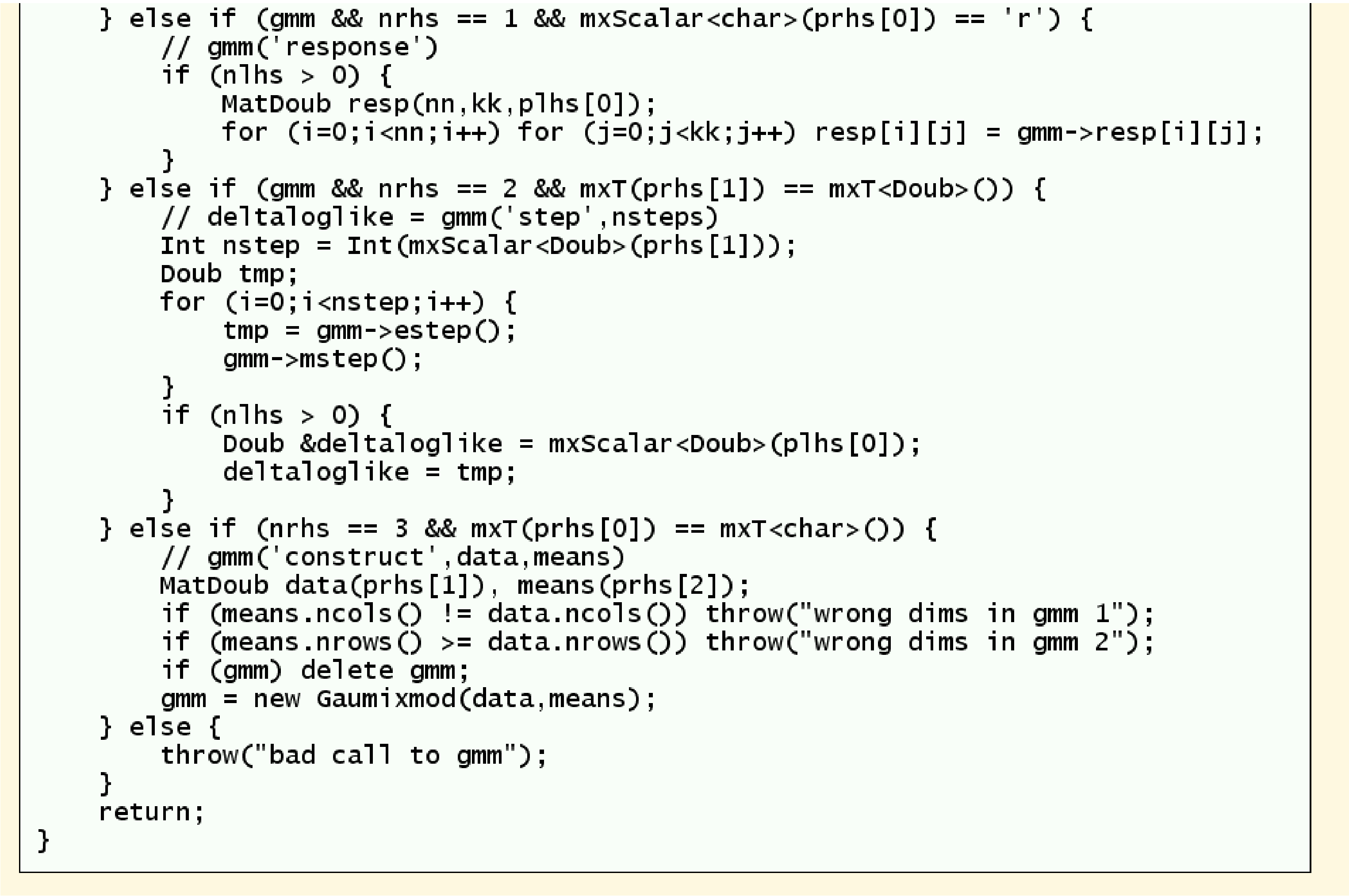


```
>> gmm('construct',data,means)   % construct the model from data and means

>> loglike = gmm('step',nsteps)  % step the model and return log-likelihood

>> [mean sig] = gmm(k) % return the mean and covariance of the kth component

>> resp = gmm('response') % return the response matrix

>> gmm('delete') % delete the model
```

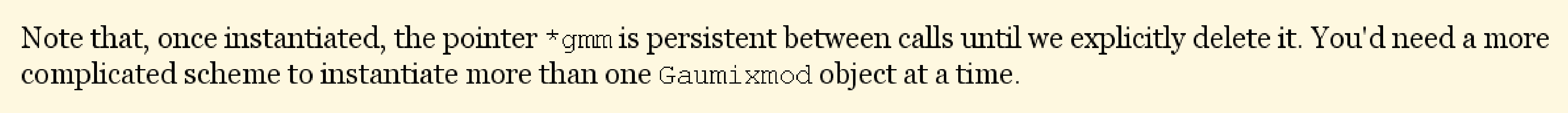


```
/* gmm.cpp */
#include "nr3matlab.h"
#include "cholesky.h"
#include "gaumixmod.h"

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];
        }
        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;
```

from: http://www.nr.com/nr3_matlab.html



```
  } 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");
    if (gmm) delete gmm;
    gmm = new Gaumixmod(data,means);
  } else {
    throw("bad call to gmm");
  }
  return;
}
```

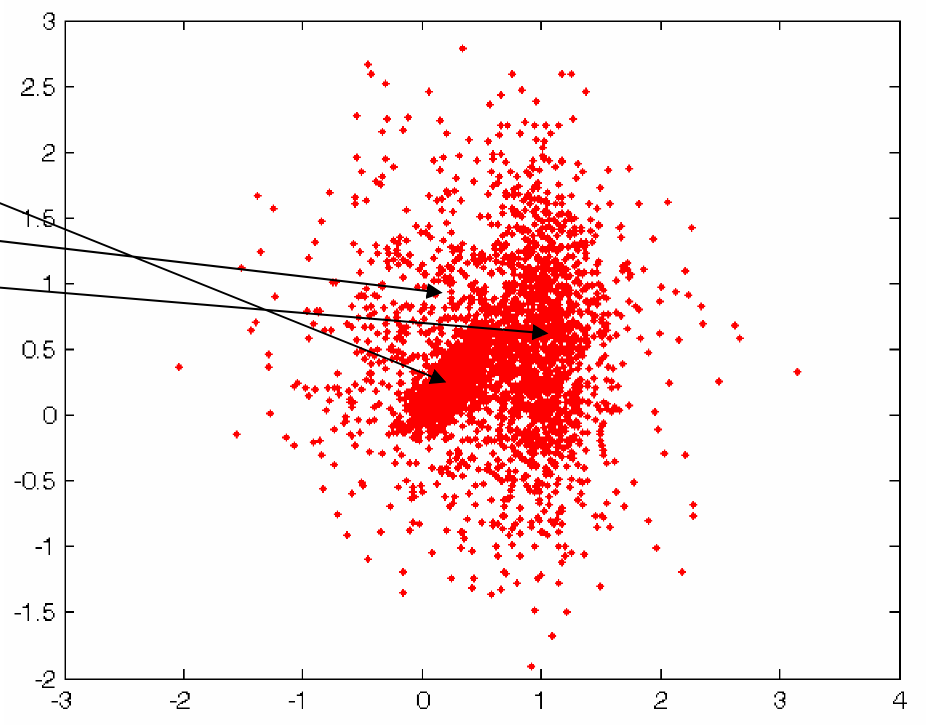


Note that, once instantiated, the pointer *gmm is persistent between calls until we explicitly delete it. You'd need a more complicated scheme to instantiate more than one Gaumixmod object at a time.

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')
```

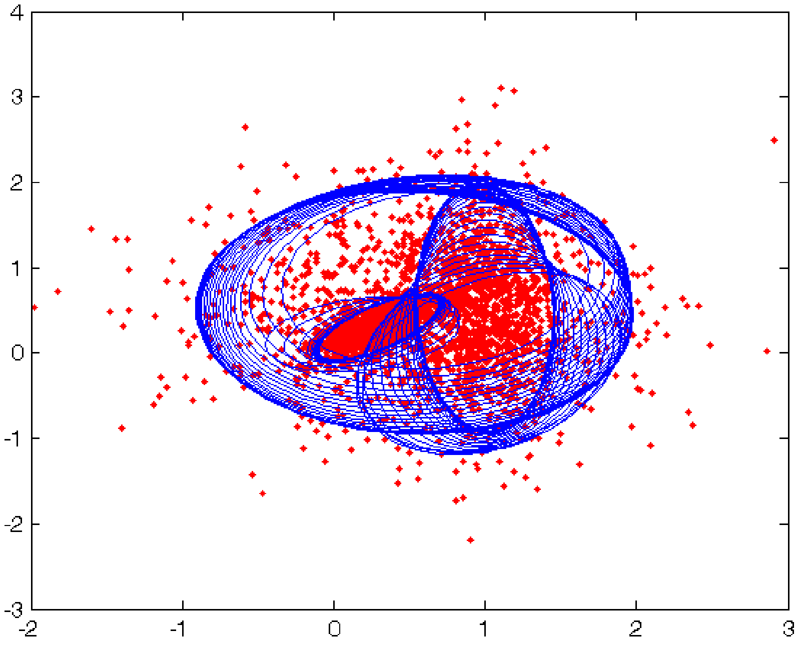


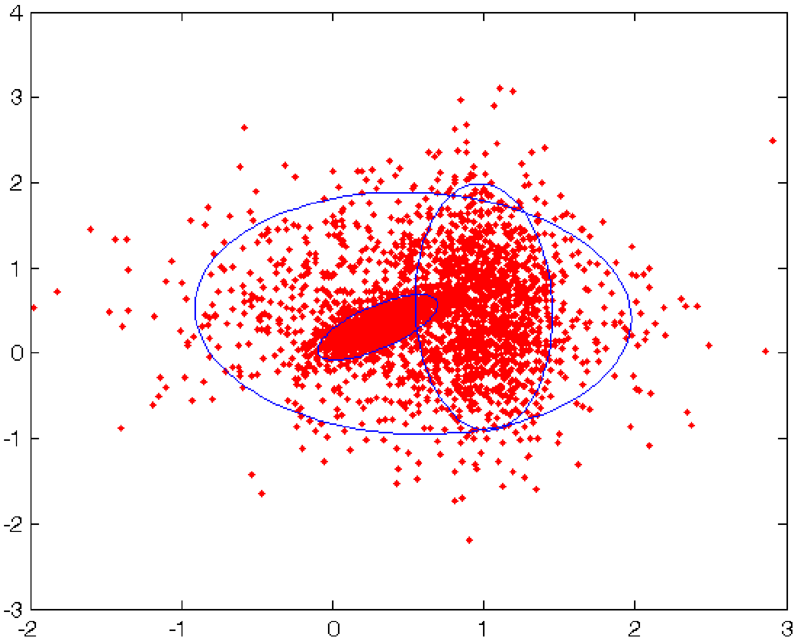
Use our mex function “gmm”:

```
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?

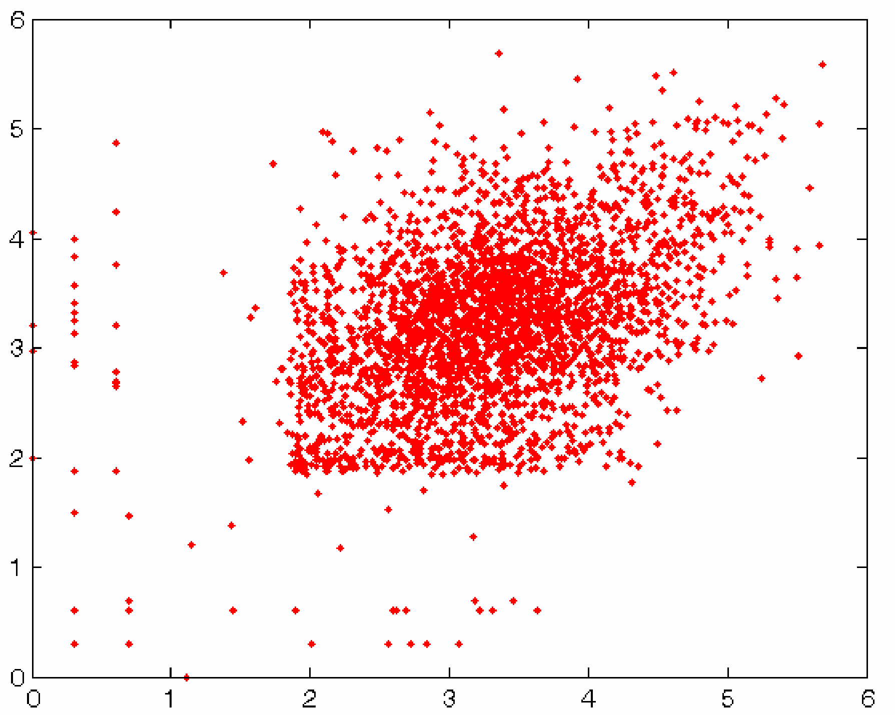




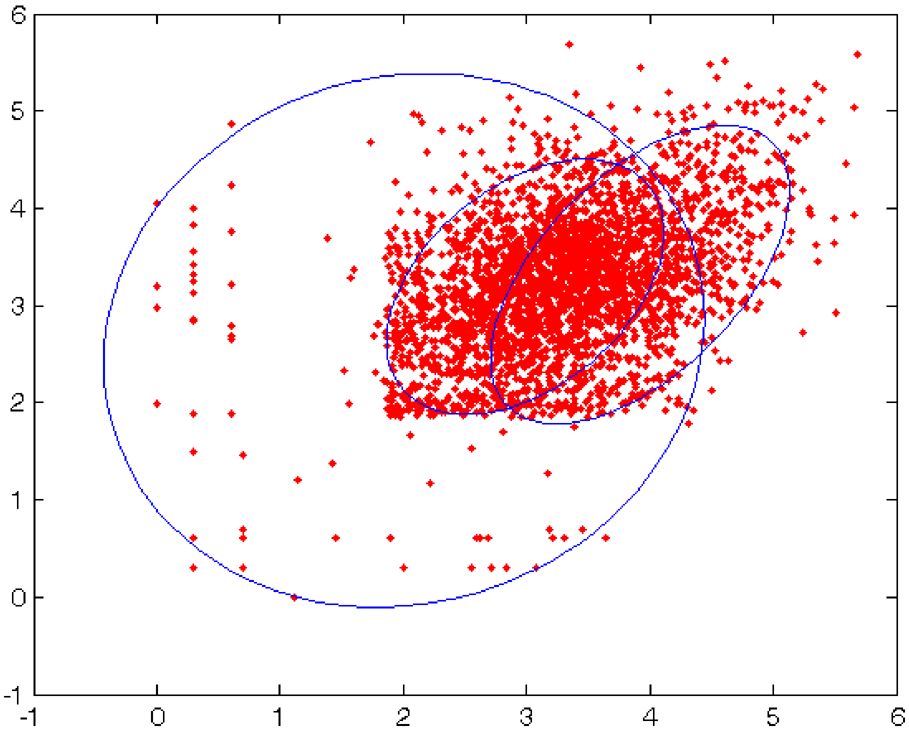
This “ideal” example converges rapidly to the right answer.

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".

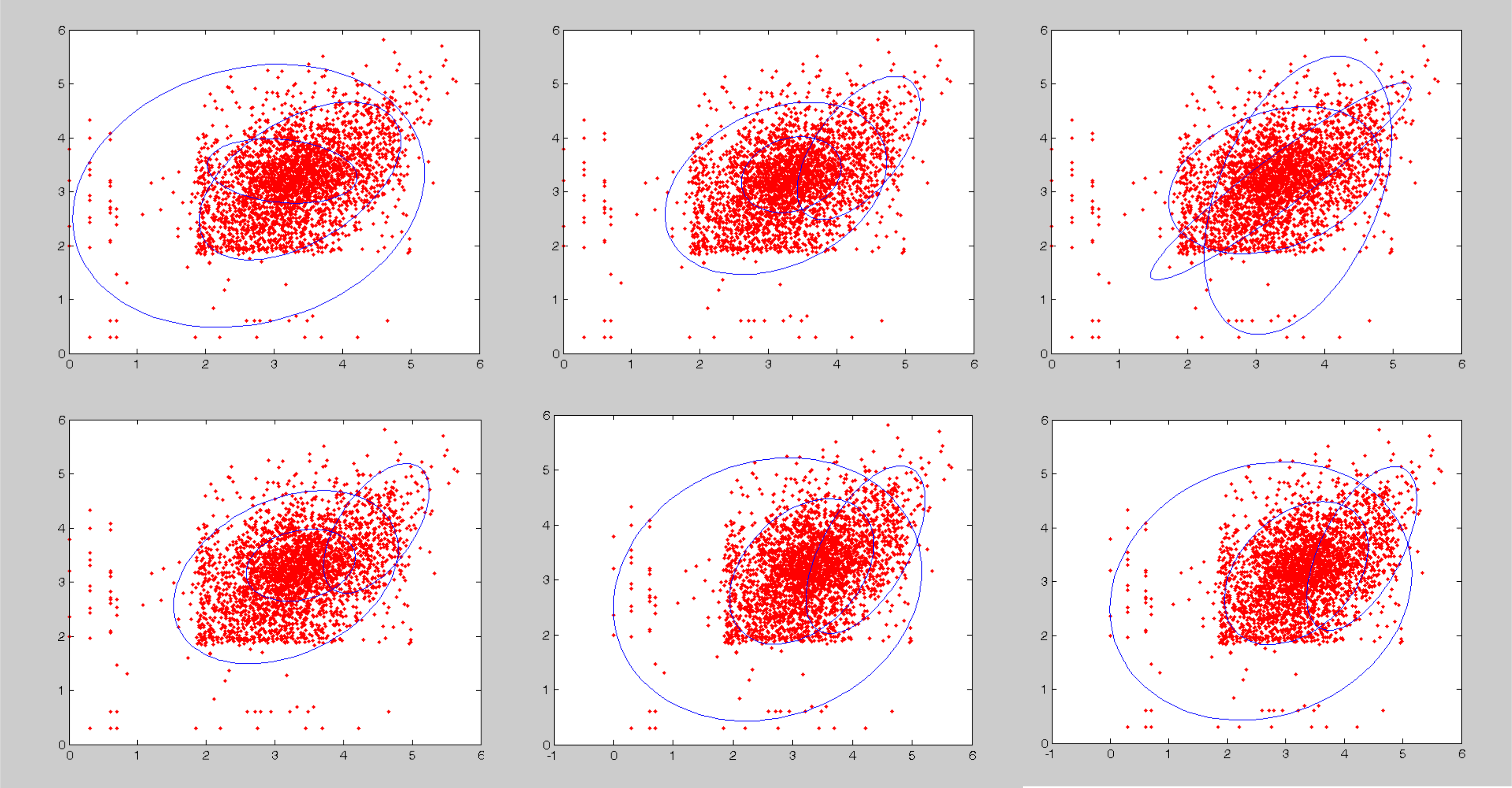
```
g = readgenestats('genestats.dat');
ggg = g(g.ne>2,:);
which = randsample(size(ggg,1),3000);
iilen = ggg.intronlen(which);
i1len = zeros(size(which));
i2len = zeros(size(which));
for j=1:numel(i1len), i1llen(j) = log10(iilen{j}(1)); end;
for j=1:numel(i2len), i2llen(j) = log10(iilen{j}(2)); end;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



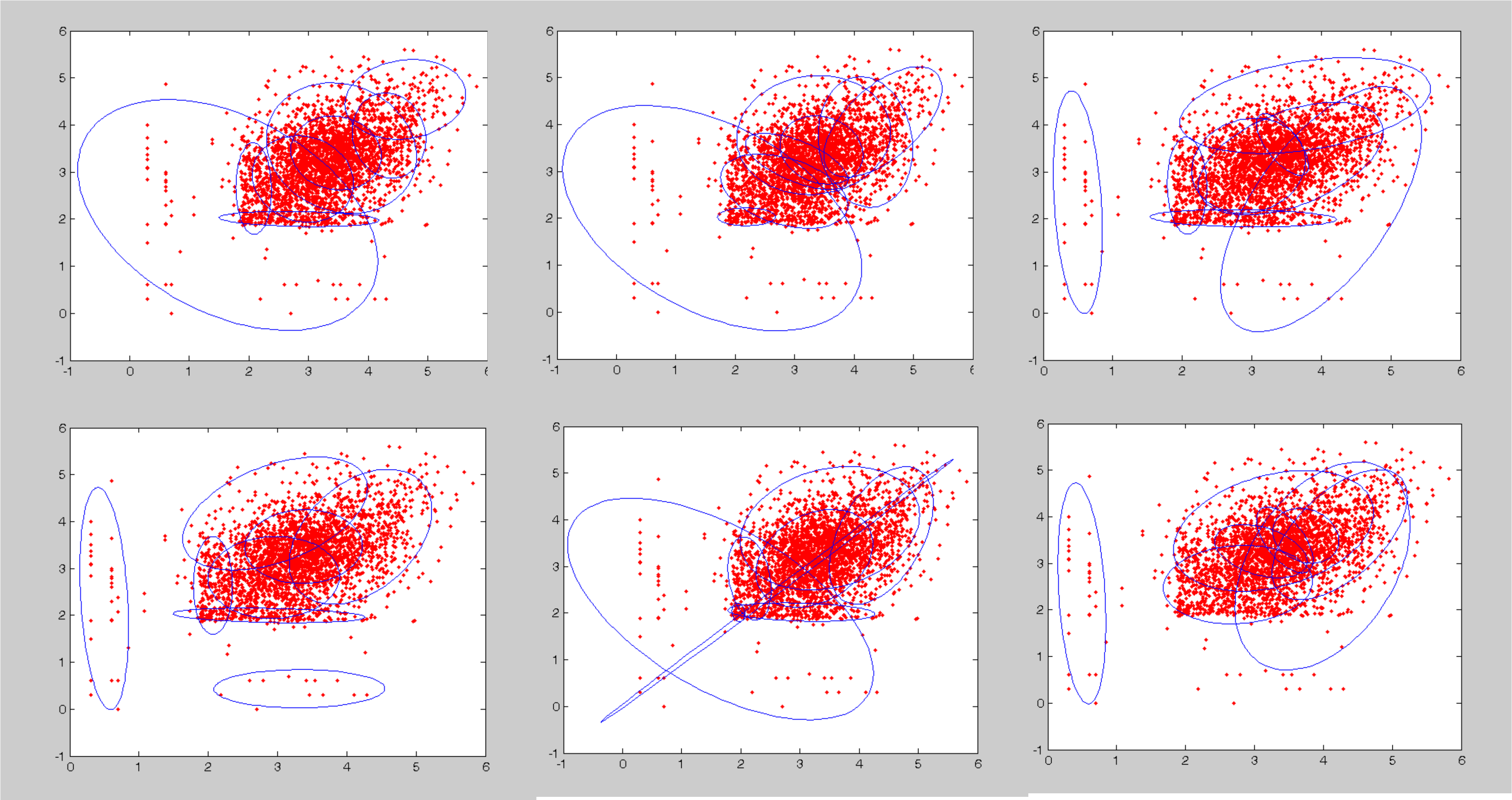
```
ncomp = 3;
plot(rsamp(:,1),rsamp(:,2),'.r')
hold on
means = zeros(ncomp,2);
for k=1:ncomp; means(k,:) = rsamp(ceil(rand*3000),:); end;
gmm('construct',rsamp',means');
deltaloglike = 1.e10;
while deltaloglike > 0.1;
    deltaloglike = gmm('step',1);
end;
for k=1:ncomp;
    [mmu ssig] = gmm(k);
  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

GMMs can have simplified models for the shapes (covariances) of components

- 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)

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

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

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

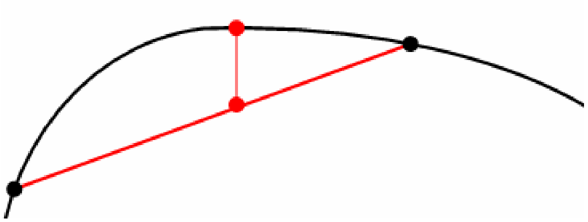
- Or fix Σ = ε **1** (infinitesimal times unit matrix)
  - don’t re-estimate Σ, only re-estimate μ
  - this assigns points 100% to the closest cluster (so don’t actually need to compute any Gaussians, just compute distances)
  - it is called “K-means clustering”
    - kind of GMM for dummies
    - widely 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

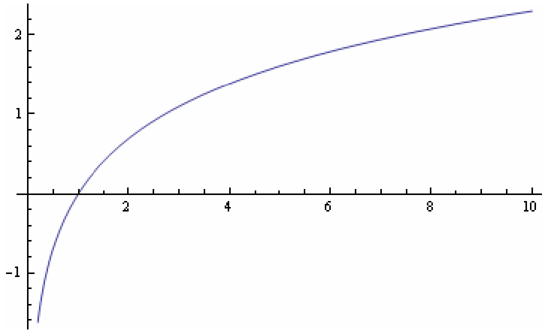
function(interpolation) ≥ interpolation(function)



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.