Opinionated Lessons in Statistics

by Bill Press

#29 GMMs in N-dimensions

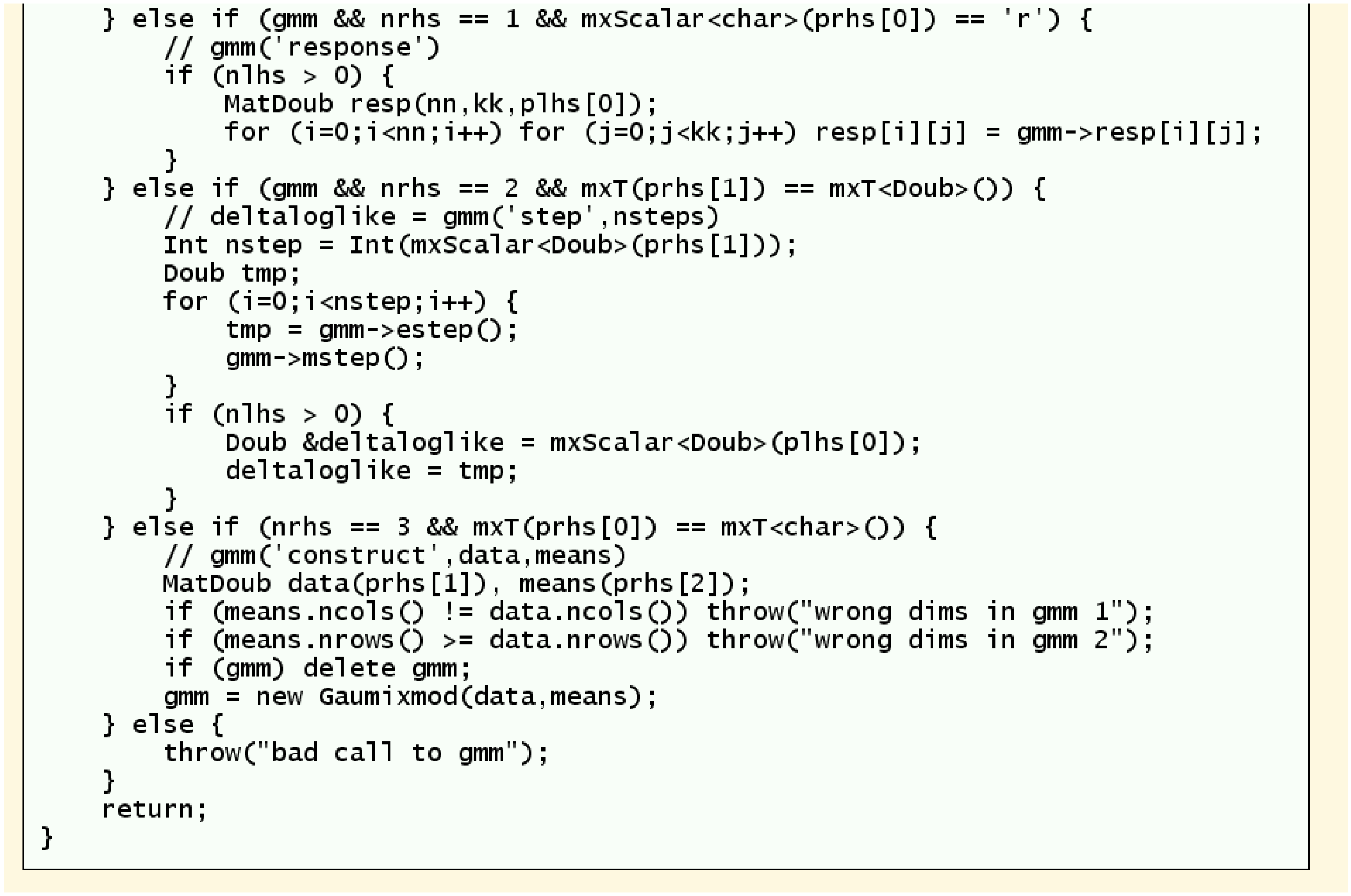
from: http://www.nr.com/nr3_matlab.html

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

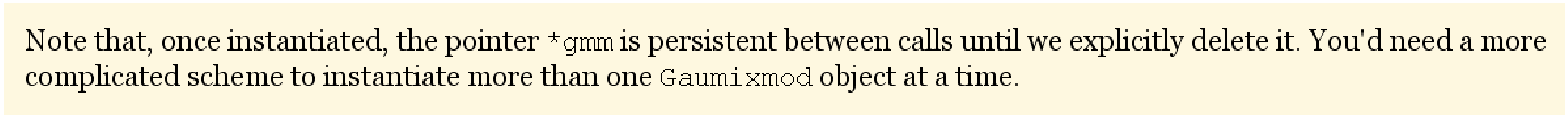
```
>> gmm('construct',data,means)   % construct the model from data and means

>> 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;
}
```

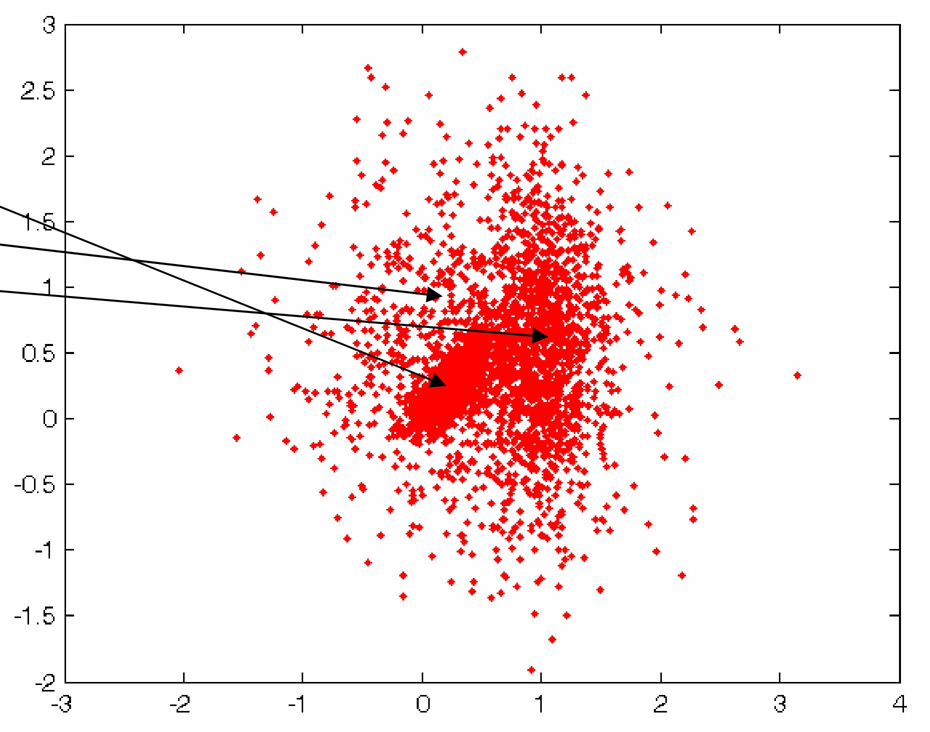


Note that, once instantiated, the pointer `*gmm` is persistent between calls until we explicitly delete it. You'd need a more complicated scheme to instantiate more than one `Gaumixmod` object at a time.

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')
```

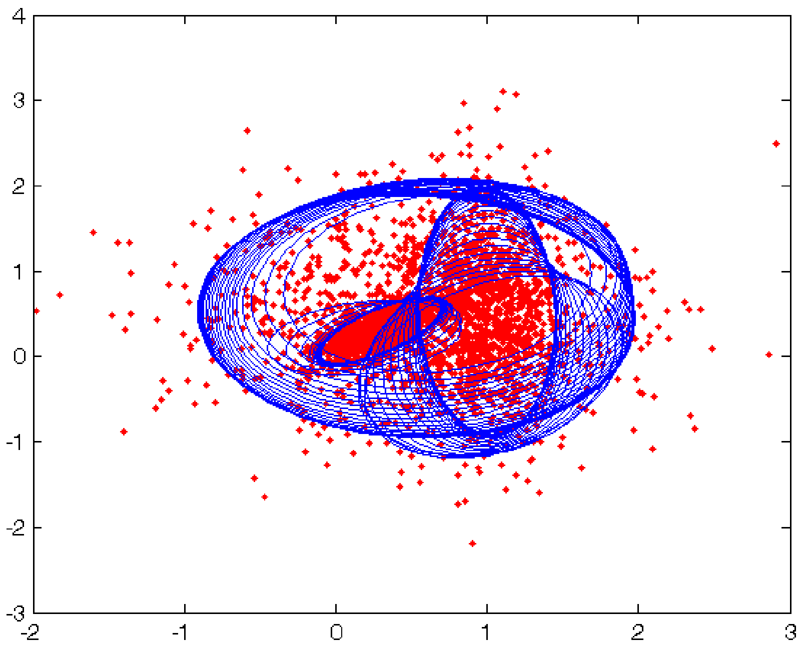


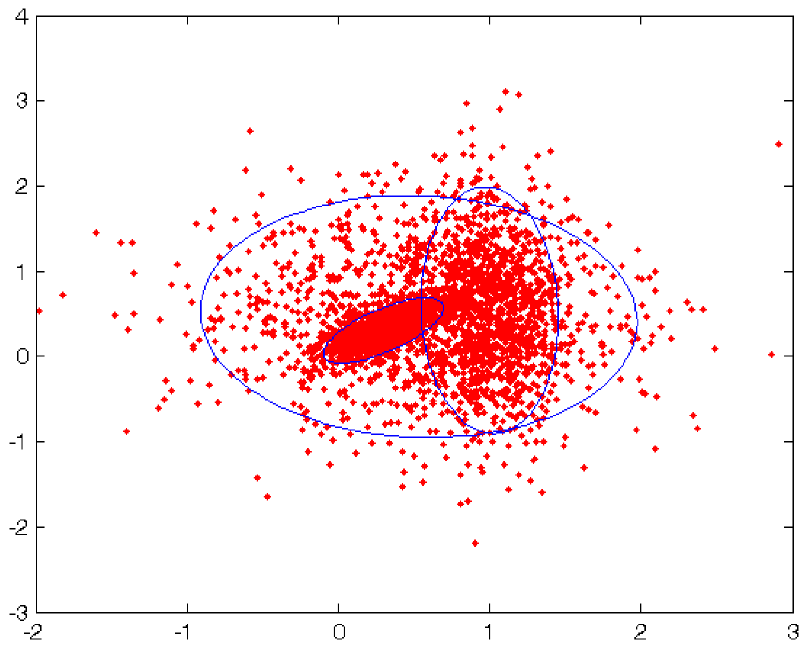
Use our mex function “gmm”:

```
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?

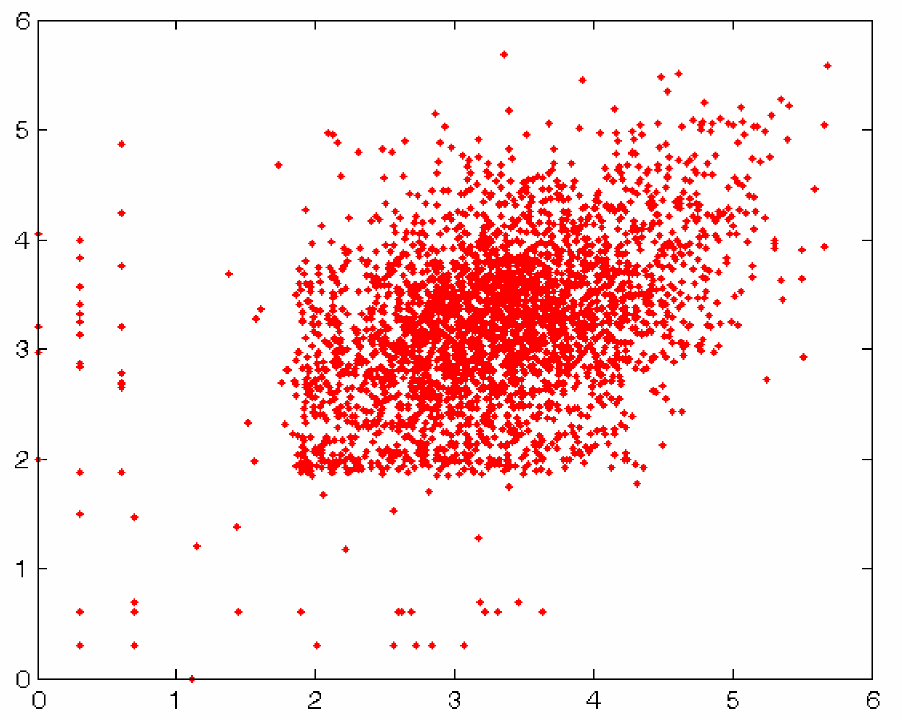




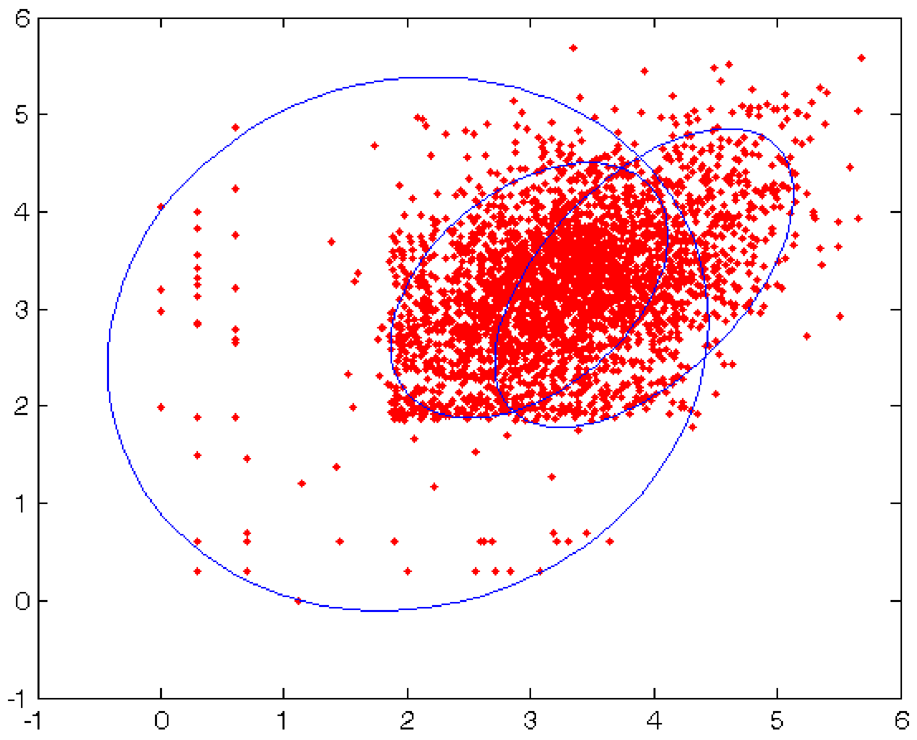
This “ideal” example converges rapidly to the right answer.

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".

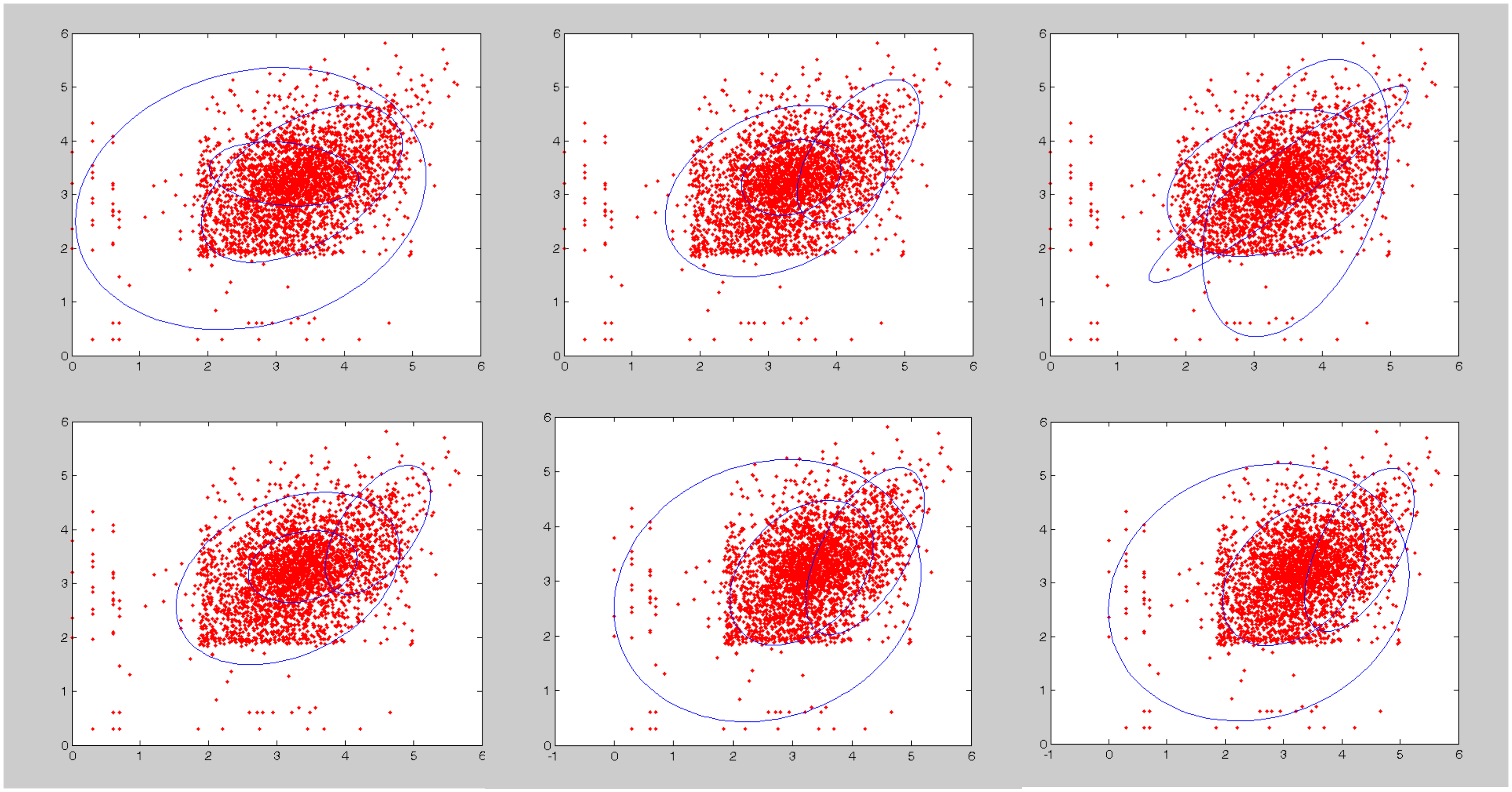
```
g = readgenestats('genestats.dat');
ggg = g(g.ne>2,:);
which = randsample(size(ggg,1),3000);
iilen = ggg.intronlen(which);
i1len = zeros(size(which));
i2len = zeros(size(which));
for j=1:numel(i1len), i1llen(j) = log10(iilen{j}(1)); end;
for j=1:numel(i2len), i2llen(j) = log10(iilen{j}(2)); end;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



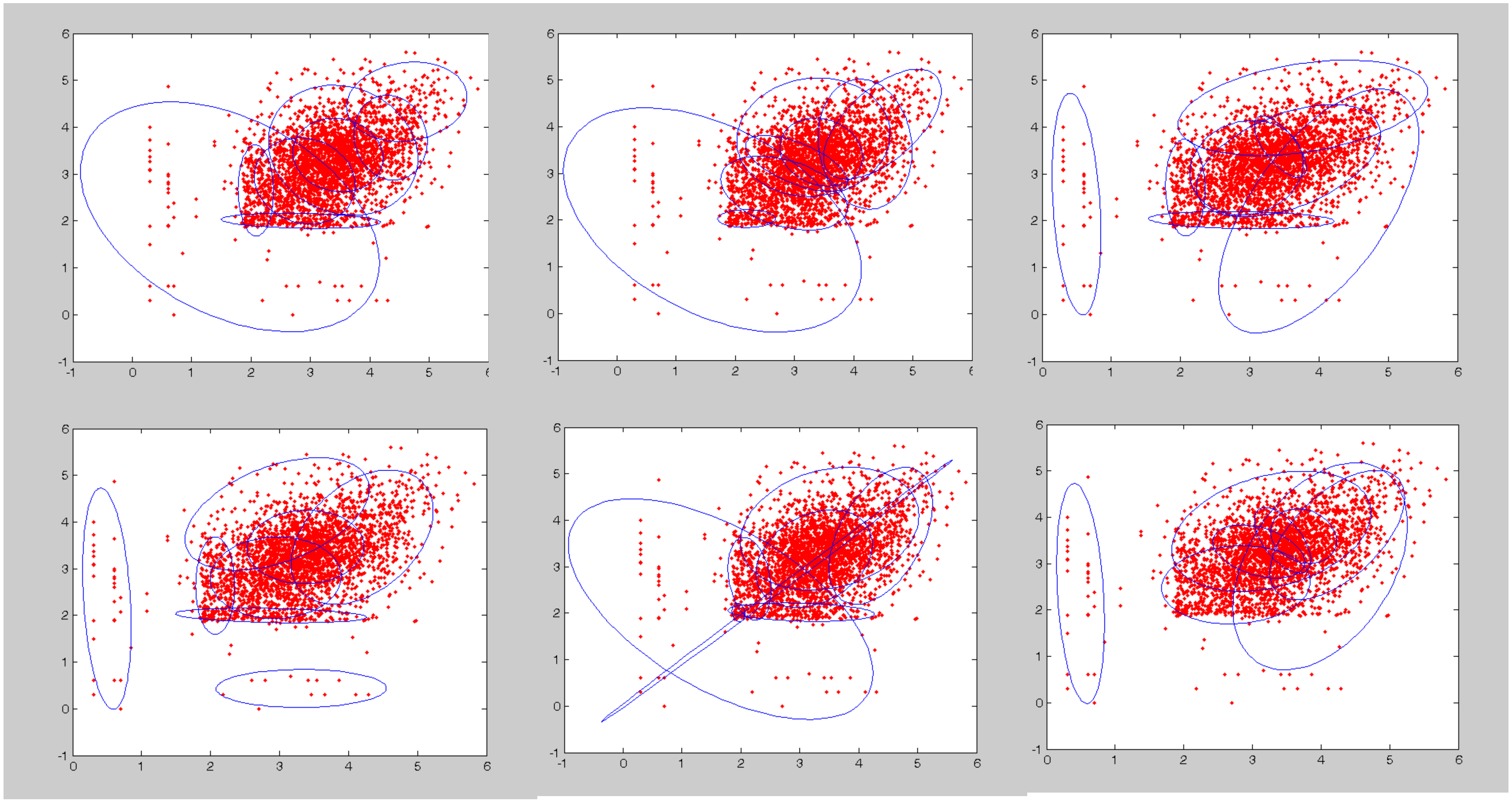
```
ncomp = 3;
plot(rsamp(:,1),rsamp(:,2),'.r')
hold on
means = zeros(ncomp,2);
for k=1:ncomp; means(k,:) = rsamp(ceil(rand*3000),:); end;
gmm('construct',rsamp',means');
deltaloglike = 1.e10;
while deltaloglike > 0.1;
    deltaloglike = gmm('step',1);
end;
for k=1:ncomp;
    [mmu ssig] = gmm(k);
  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