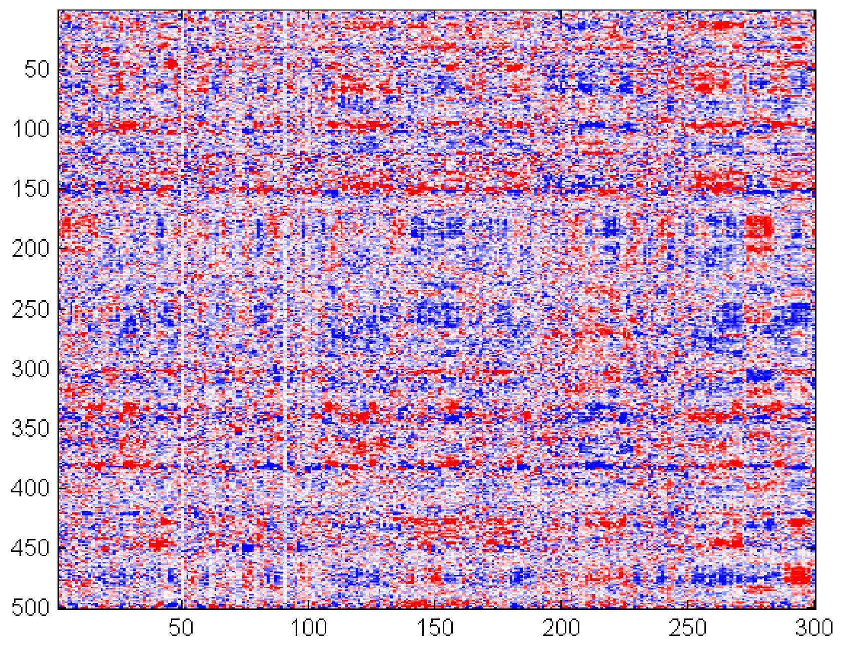
CS395T
Computational Statistics with Application to Bioinformatics

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

Lecture 27

NEW TOPIC: Linear "Stuff": SVD, PCA, Eigengenes, and all that!

We start with a "data matrix" or "design matrix": $\mathbf{X} = \{X_{ij}\}$
Let's use gene expression data as the example.



(This is just to give you something to look it. Typically, for these techniques, you would not be able to see anything in the data "by eye".)

N rows are data points, here genes 1 – 500
M columns are the responses. View each as a vector in M (here 300) dimensional space
For gene expression data, each column is a separate micro array experiment under a different condition

Let's always assume that the individual experiments (columns of $\mathbf{X}$) have zero mean. (Can always get this by subtracting the mean of each column.)

While we're at it, we might as well also scale each column to unit standard deviation.

And, it's a good idea to eliminate outliers.

Matlab for this (and plotting the previous picture) is:

```
load yeastarray_t2.txt;
size(yeastarray_t2)
ans =
   500   300
yclip = prctile(yeastarray_t2(:),[1,99])
yclip =
  -204   244
data = max(yclip(1),min(yclip(2),yeastarray_t2));
dmean = mean(data,1);
dstd = std(data,1);
data = (data - repmat(dmean,[size(data,1),1]))./repmat(dstd,[size(data,1),1]);
genecolormap = [min(1,(1:64)/32); 1-abs(1-(1:64)/32); min(1,(64-(1:64))/32)]';
colormap(genecolormap);
image(20*data+32)
```

clip outliers by percentile (bottom and top 1%)

this is the arcane Matlab colormap stuff for making a blue-to-white-to-red plot that we saw before

saturate to red or blue at ~1.5 σ

Singular Value Decomposition (SVD)

Any matrix $\mathbf{X}$ (needn’t be square) can be decomposed, more-or-less uniquely, as follows:

$$\begin{pmatrix} \\ \\ \mathbf{X} \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ \mathbf{U} \\ \\ \\ \end{pmatrix} \begin{pmatrix} s_1 & & \\ & s_2 & \\ & & \cdots \end{pmatrix} \begin{pmatrix} \mathbf{V}^T \end{pmatrix}$$

rows (cols of V) are a complete orthonormal basis for M dimensional space

diagonal matrix of positive “singular values” arranged from largest to smallest

cols are orthonormal basis for an M dimensional subspace of N

Degree of freedom (DOF) counting:

$$MN \quad = \quad MN - M(M+1)/2 \quad + M \quad + M^2 - M(M+1)/2$$

Decomposition has an efficient algorithm (of order the same workload as inverting a matrix). Matlab and NR3 have ready-to-use implementations.

We can write out the (middle) sums over the singular values explicitly. Each column of $\mathbf{U}$ gets paired with the corresponding row of $\mathbf{V}^T$ (or column of $\mathbf{V}$).

$$\mathbf{X} = \sum_{i=1}^{M} s_i \, \mathbf{U}_{\cdot i} \otimes \mathbf{V}_{\cdot i}$$

note: "dot" now does NOT mean sum. It's just a placeholder!

This turns out to be the optimal decomposition of $\mathbf{X}$ into rank-1 matrices, optimal in the sense that the partial sums converge in the "greediest" way in L^2 norm. (I.e., at each stage in the sum, there is no better decomposition.)

Recall: A rank-one matrix has all its columns proportional to each other, which also implies that all the rows are proportional to each other: $C_{ij} = A_i \, B_j$ So the rows (or columns) lie on a one-dimensional line in the row (or column) dimension space.

$$\begin{aligned}
\sum_i \sum_j |X_{ij}|^2 = \mathrm{Tr}(\mathbf{X}\mathbf{X}^T) &= \sum_i \left[\sum_j X_{ij} X^T_{ji} \right] \\
&= \mathrm{Tr}(\mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V}\mathbf{S}\mathbf{U}^T) \\
&= \mathrm{Tr}(\mathbf{U}\mathbf{S}^2\mathbf{U}^T) \\
&= \mathrm{Tr}(\mathbf{S}^2\mathbf{U}^T\mathbf{U}) \\
&= \mathrm{Tr}(\mathbf{S}^2)
\end{aligned}$$

$$\mathbf{X} = \sum_{i=1}^{M} s_i \, \mathbf{U}_{\cdot i} \otimes \mathbf{V}_{\cdot i}$$

So this "builds up dimensionality" with each term in the sum: adds a new basis vector (in both the U and the V spaces)

If the data actually lie on a lower dimensional (than M) hyperplane that goes through the origin, then only that many s_i's will be nonzero.

That is why we subtracted the means!

Or, in the general case we can just truncate the sum to get the best lower rank approximation to $\mathbf{X}$. This can be useful for filtering out noise (we will see).

Notice that this captures only a "linear" (hyperplane) view of the world. More complicated functional relationships that might decrease dimensionality are not, in general, identified by SVD or PCA.

Principal Component Analysis (PCA)

Note that the (sample) covariance of the experiments is:

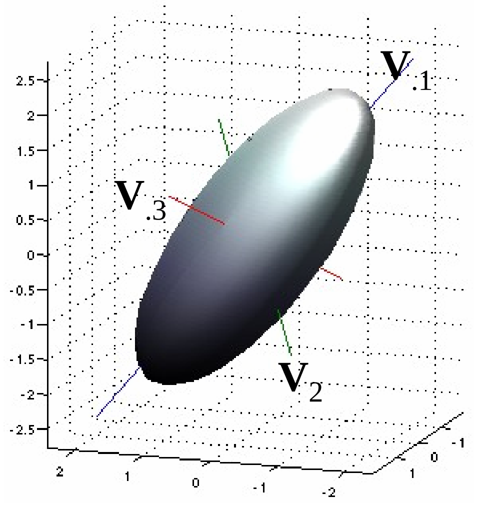
$$\mathrm{Cov}(\mathrm{Expt}_i, \mathrm{Expt}_j) = \Sigma_{ij} = \frac{1}{N}\sum_k X_{ki}X_{kj}$$

Uses fact that we subtracted the means : ‹ x ⊗ x ›.

$$N\mathbf{\Sigma} = \mathbf{X}^T\mathbf{X} = (\mathbf{V}\mathbf{S}^T\mathbf{U}^T)(\mathbf{U}\mathbf{S}\mathbf{V}^T) = \mathbf{V}(\mathbf{S}^2)\mathbf{V}^T$$

diagonal

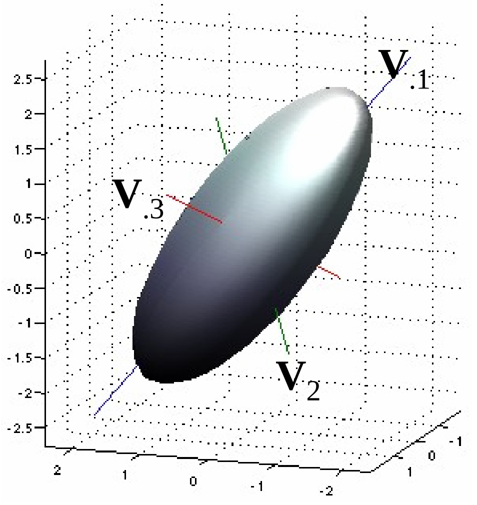
So V is a rotation matrix that diagonalizes the covariance matrix.



in our example, 300 dim space with 500 scattered points in it – and now a covariance ellipsoid

It follows that the data points in $\mathbf{X}$ have their largest variance in the $\mathbf{V}_{.1}$ direction.
Then, in the orthogonal hyperplane, the 2nd largest variance is in the $\mathbf{V}_{.2}$ direction.
And so forth.

So we might usefully coordinatize the data points by their M projections along the $\mathbf{V}_{.i}$ directions (instead of their M raw components). These projections are a matrix the same shape as $\mathbf{X}$. Since the directions are orthonormal columns, it is simply



$$\mathbf{XV} = \mathbf{US} \qquad (\text{using } \mathbf{X} = \mathbf{USV}^T)$$

rows are the data points,
column components are the principal coordinates of that row

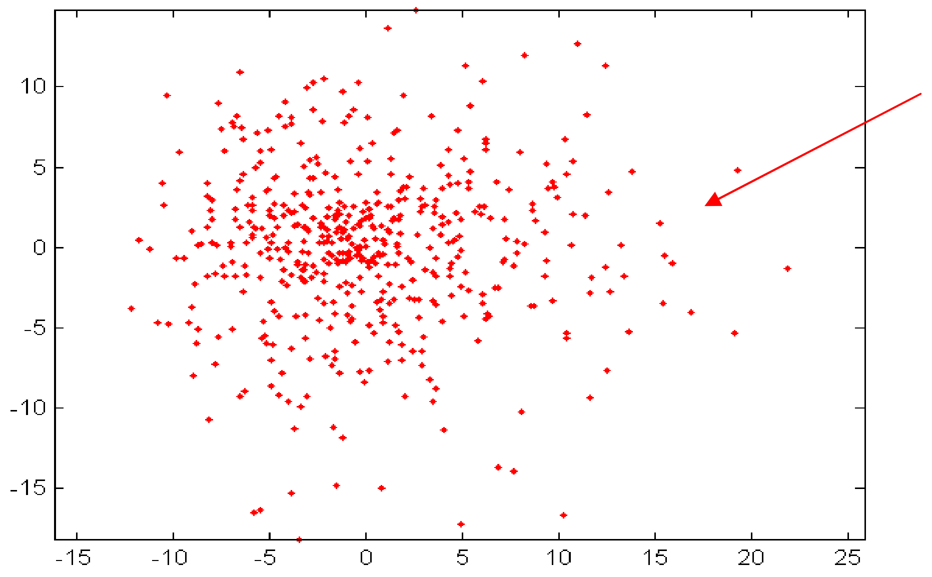
Also, it's easy to see that (by construction) the principal components of the points are uncorrelated, that is, have a diagonal correlation matrix:

diagonal

$$(\mathbf{XV})^T(\mathbf{XV}) = (\mathbf{US})^T(\mathbf{US}) = \mathbf{S}^T(\mathbf{U}^T\mathbf{U})\mathbf{S} = \mathbf{S}^2$$

Lets plot our expression data in the plane of the top 2 principal components:

```
[U S V] = svd(data, 0);
pcacoords = U*S;
plot(pcacoords(:,1),pcacoords(:,2),'r.')
axis equal
```

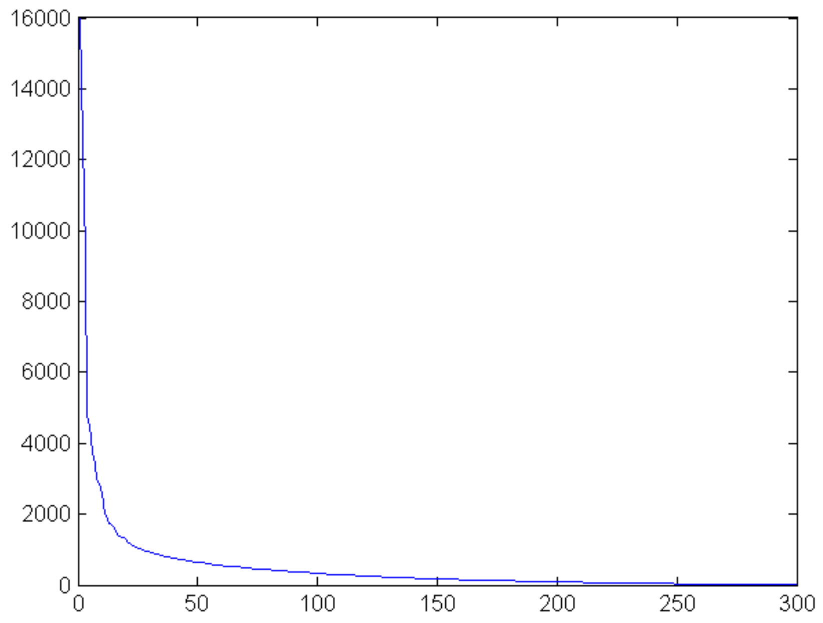


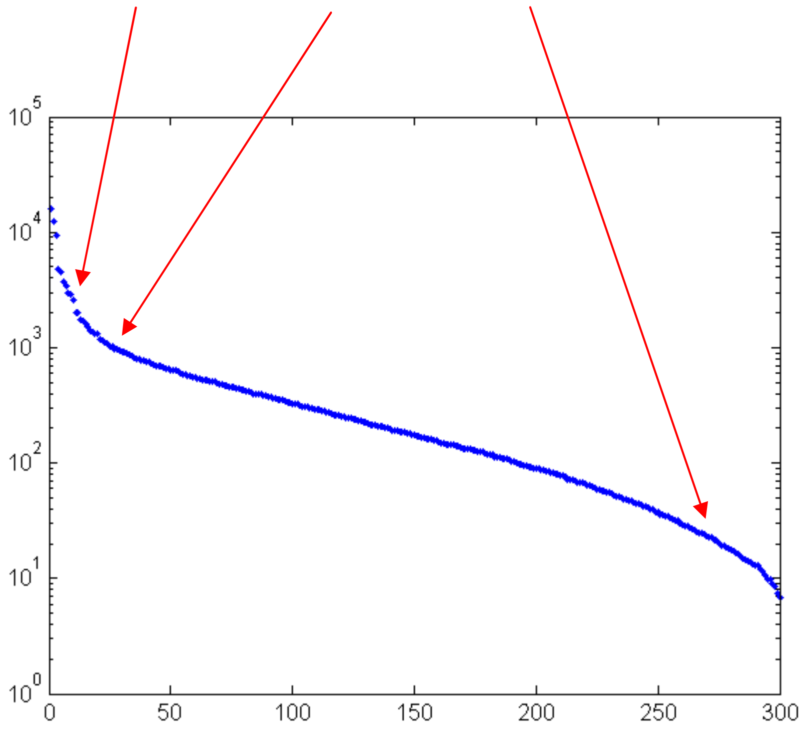
Direction 1 has larger variance than direction 2, and there is no correlation between the two, all as advertised.

As already shown, the squares of the SV's are proportional to the portion of the total variance (L^2 norm of $\mathbf{X}$) that each accounts for.

```
ssq = diag(S).^2;
plot(ssq)
semilogy(ssq,'.b')
```

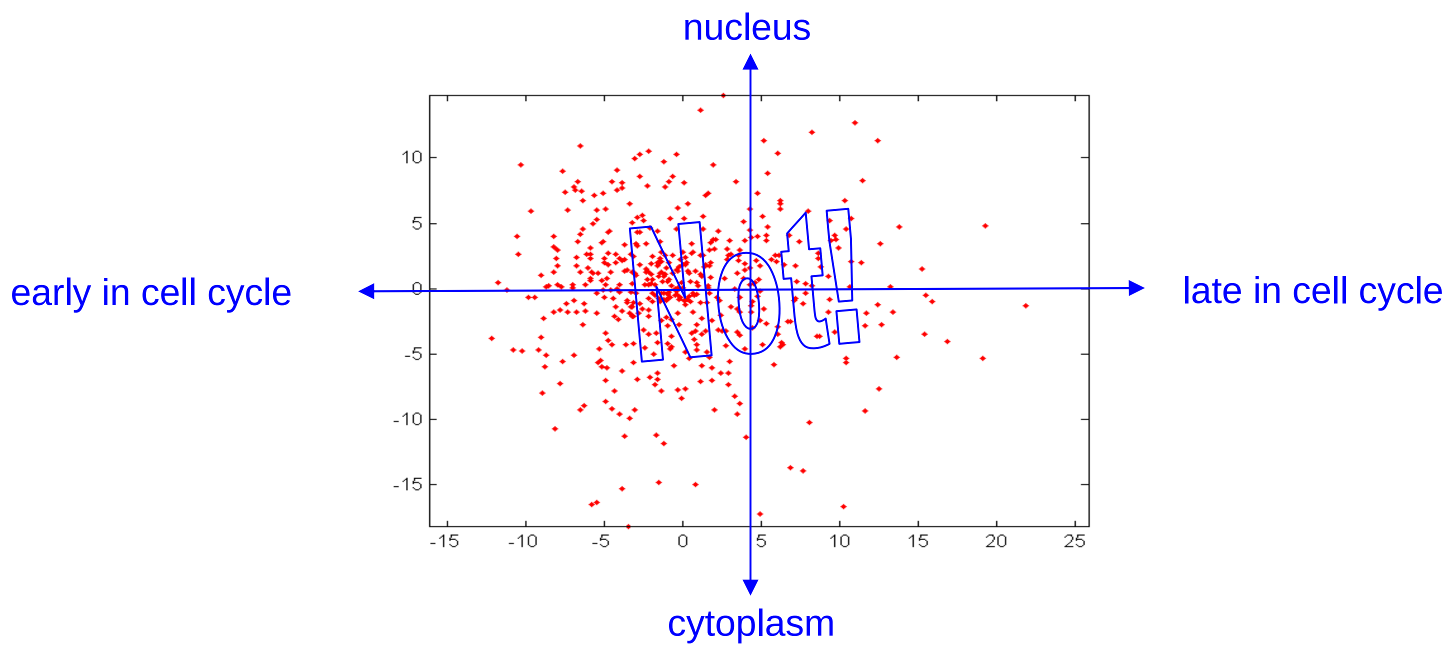
Where do these values start to be explainable simply as noise? Here? or here? or here?





(Of course, we'll never really know from a single data set, since, in the limit of fine-grain effects, signal is just repeatable noise!) But we can often make a guess after examining the data.

People who love PCA (I call them “linear thinkers”) always hope that the principal coordinates will magically correspond to distinct, real effects (“main effects”).



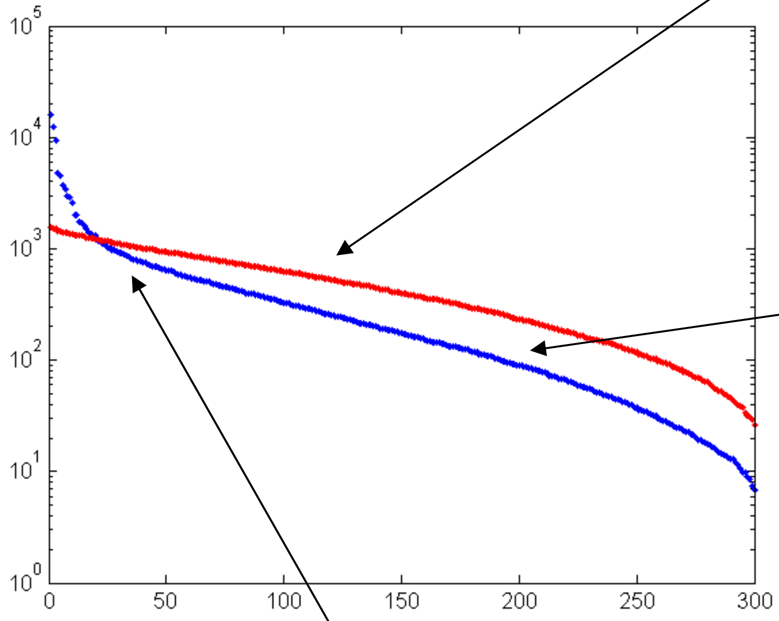
This is <u>sometimes</u> true for the 1st principal component, and <u>rarely</u> true after that. I think the reason is that orthogonality (in the mathematical sense of SVD) is rarely a useful decomposition of “distinct, main effects”, which tend to be highly correlated mathematically, even when they are “verbally orthogonal”.

However, it <u>is</u> often true that ~K main effects are captured (somewhere) in the subspace of the first ~K principal components.

So, PCA is a useful technique for dimensional reduction. Just don’t try to [over]interpret the meaning of individual coordinates! (Let’s see examples.)

One way to inform our guess as to what is signal (vs. noise) is to compare to a matrix of Gaussian random deviates:

```
fakedata = randn(500,300);
[Uf Sf Vf] = svd(fakedata,0);
sfsq = diag(Sf).^2;
semilogy(sfsq,'.r')
```



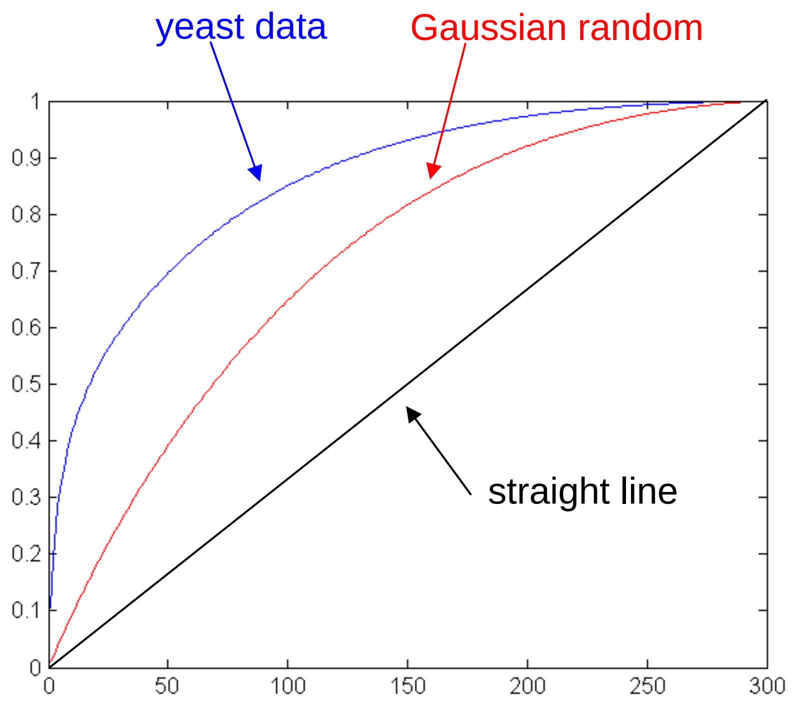
Why does fake show a trend at all? Because even random numbers are monotonic if you sort them! We are seeing the “order statistics” for SVs from a Gaussian random matrix.

Fake has to be higher than real here, because area under the curves (if they were plotted on a linear scale) has to be the same for the two curves (same total variance or L^2 norm)

I’d say it’s questionable that there’s much information in our real data set beyond around here

Sometimes, people plot the fractional variance as a function of number of SVs, which also shows how we converge to an exact SV decomposition:

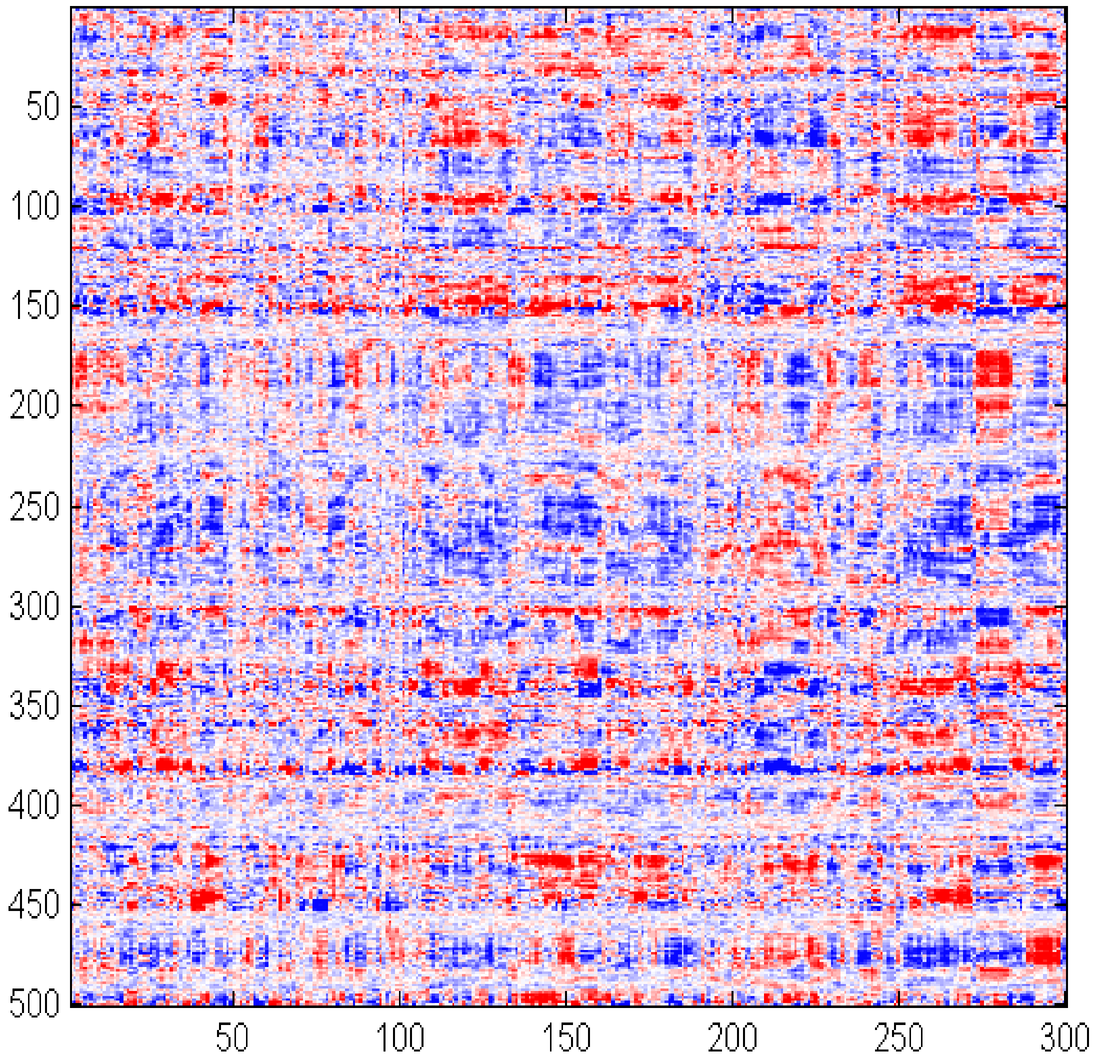
```
ssqnorm = cumsum(ssq)/sum(ssq);
sfsqnorm = cumsum(sfsq)/sum(sfsq);
plot(ssqnorm,'b')
hold on
plot(sfsqnorm,'r')
hold off
```



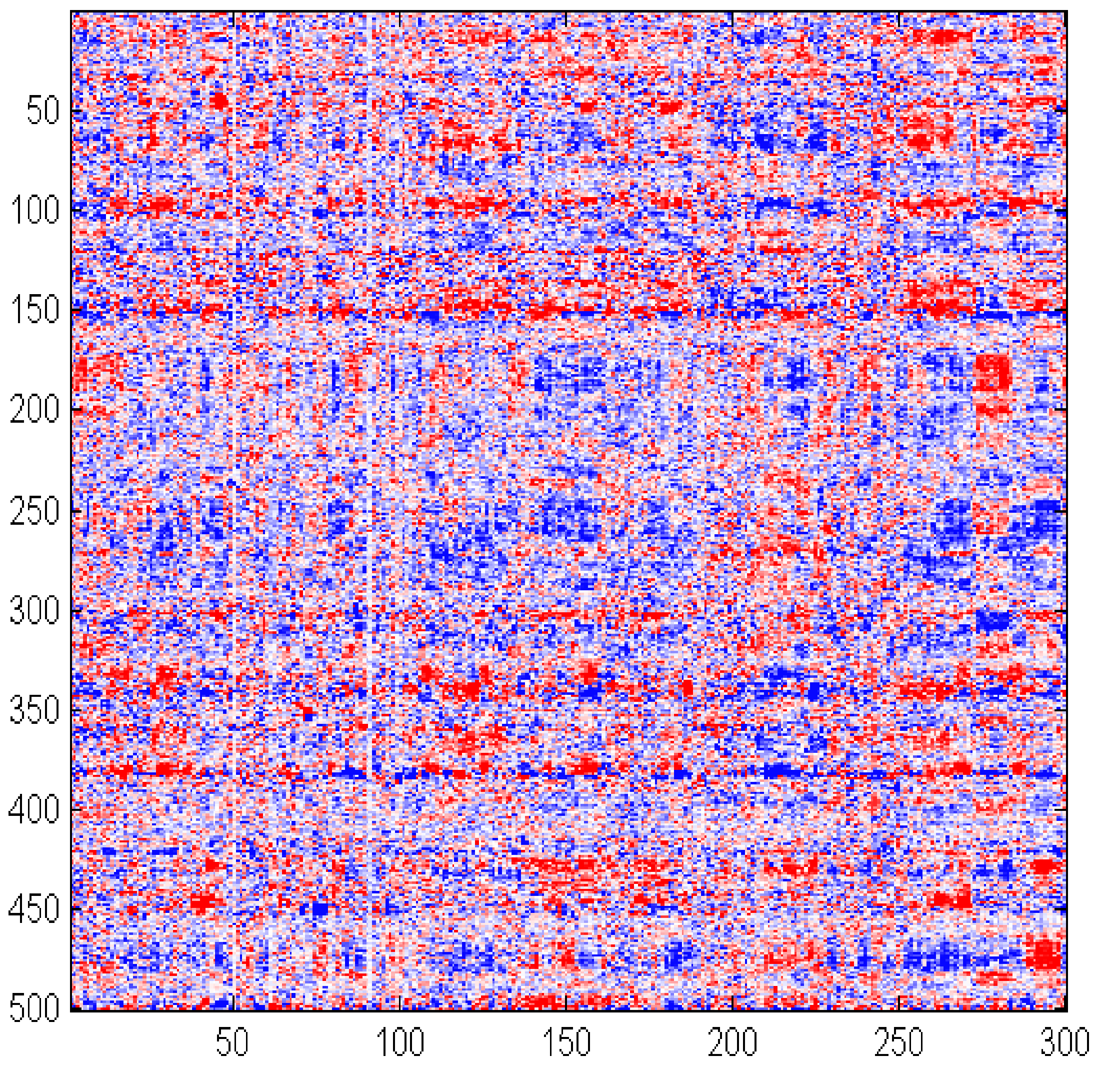
You might have expected the Gaussian random to be close to the straight line, since each random SV should explain about the same amount of variance. But, as before, we're seeing the (sorted) order statistics effect. So it is actually rather hard to interpret from this plot (if you had only the blue curve) what is real versus noise and how impressed you should be by a rapid initial rise.

For the data in this example, a sensible use of PCA (i.e., SVD) would be to project the data into the subspace of the first ~20 SVs, where we can be sure that it is not noise.

```
strunc = diag(S);
strunc(21:end) = 0;
filtdata = U*diag(strunc)*V';
colormap(genecolormap);
image(20*filtdata+32)
```

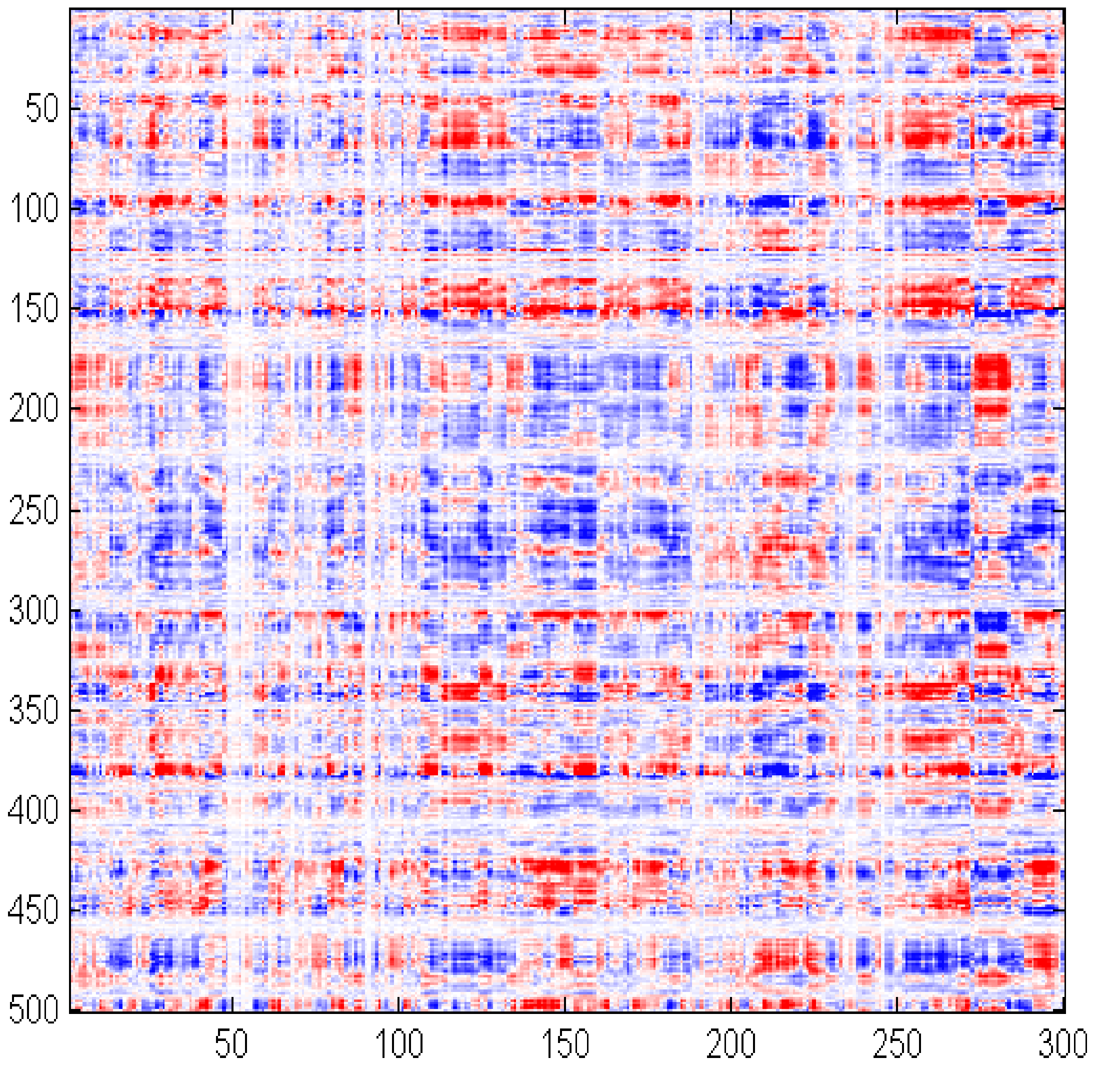


original data set:



Or, just 5 SV's:

```
strunc(6:end) = 0;
filtdata = U*diag(strunc)*V';
colormap(genecolormap);
image(20*filtdata+32)
```


50
100
150
200
250
300
350
400
450
500
50
100
150
200
250
300